VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Racionalizace zakázkového systému ve stavebnictví

Rationalization of Order System in the Building Industry

Student: Jiří Toman

Vedoucí diplomové práce: Ing. Vítězslav Novák, Ph.D.

Ostrava 2010

„Místopřísežně prohlašuji, že jsem celou práci včetně všech příloh vypracoval samostatně“

.....

Obsah

1. Úvod

Téma diplomové práce „Racionalizace zakázkového systému ve stavebnictví“ jsem si zvolil především z toho důvodu, že jsem chtěl realizovat své dosavadní znalosti a dovednosti, které jsem během studia na VŠB-TU Ostrava získal. Zároveň jsem chtěl vytvořit něco konkrétního. Něco, co by mělo přínos v reálném světě.

Cílem diplomové práce je racionalizace zakázkového systému konkrétní firmy. Pro tyto účely jsem zvolil firmu Telko s r.o., která mimo jiné podniká v oblasti stavebních prací a projevila o vzájemnou spolupráci zájem. Při rozhodování o výběru firmy hrál hlavní roli fakt, že prostředí firmy Telko s r.o. již delší dobu znám. Myslím si, že jsem udělal dobře, když jsem si pro aplikaci svých znalostí vybral právě firmu Telko s r.o.

Diplomovou práci jsem rozčlenil do čtyř hlavních částí. První kapitola s názvem „Analýza stávajícího systému“ bude zaměřena na teoretická východiska dané problematiky a na analýzu současného systému. Již dopředu bych se rád zmínil, že současný evidenční systém má řadu slabin. Primárním cílem diplomové práce bude tedy vytvoření nového systému, který odstraní nežádoucí slabiny systému stávajícího.

Ve druhé kapitole s názvem „Návrh databázové aplikace“ nejprve uvedu svou vizi budoucího systému evidence zakázek. Následovat bude specifikace uživatelských požadavků, návrh datové struktury, návrh a popis klíčových komponent aplikace, návrh grafického uživatelského rozhraní aplikace a nakonec pojednání o možnostech portability aplikace.

Obsah třetí kapitoly, kterou jsem pojmenoval „Implementace databázové aplikace v programovacím jazyce Java“, zaměřím zpočátku na stručný popis použitých technologií MySQL a Java. Následně se pomocí vybraných fragmentů aplikačního kódu pokusím demonstrovat implementaci programovacího jazyka Java. Předvedu implementaci navázání konektivity s databázovým serverem MySQL, práci s komponentou javax.swing.JTable a implementaci knihovny JExcelAPI, jež není součástí standardních knihoven Javy.

V poslední kapitole s názvem „Zhodnocení přínosů“ se pokusím na základě několika klíčových faktorů srovnat systém původní a systém nově vyvinutý. Poté bych chtěl ještě věnovat několik vět možnému budoucímu rozvoji aplikace. Toto je vize mé diplomové práce.

2. Analýza stávajícího systému

V této kapitole se postupně zaměřím na stručný popis firmy Telko s r.o., dále na teoretická východiska dané problematiky (informační systém, software z pohledu měření kvality, softwarové inženýrství, metodiky vývoje softwaru, UML, ER modelování) a nakonec na samotný popis současného systému zakázek.

2.1 Stručný popis firmy Telko s r.o.

Společnost Telko s r.o. byla založena v roce 1993 se zaměřením na dodávky a montáž koncových telekomunikačních zařízení, včetně příslušných rozvodů. Od svého vzniku firma vybudovala několik set kilometrů přístupových kabelových sítí a několik tisíc telefonních přípojek po celém území Moravy a Slezska.

Klíčovými produkty firmy Telko s r.o. jsou realizace a správa počítačových sítí, servis a dodávka registračních pokladen, realizace bezpečnostních systémů, realizace telefonních ústředen a dodávky stavebních prací pro společnost Telefónica O2 Czech Republic, a.s. (O2), na které se dále v rámci diplomové práce zaměřím.

V minulosti firma Telko s r.o. dodávala stavební práce přímo pro společnost s nynějším názvem O2. V současné době je firma Telko s r.o. subdodavatelem společnosti O2, z teritoriálního hlediska operuje na území okresu Vsetín a dodává práce pro firmu SMT s r.o., která jakožto vítěz výběrového řízení vyhlášené společností O2 spravuje oblast několika krajů včetně kraje Zlínského a Moravskoslezského. Celá situace je poměrně složitá a do zakázkového systému vnáší řadu administrativních činností.

2.2 Teoretická východiska problematiky

Předtím než začnu se samotnou analýzou současného zakázkového systému firmy Telko s r.o., bych se napřed rád zaměřil na teoretická východiska související s danou problematikou. Postupně bych chtěl v této části diplomové práce popsat pojmy informační systém, softwarové inženýrství, proces a metodiky vývoje softwaru, UML a ER modelování. Tato teoretická východiska chápu jako základ k dalším kapitolám diplomové práce, proto si myslím, že je nezbytné se o nich alespoň zmínit.

2.2.1 Informační systém (IS)

Pokud budeme hovořit o informačním systému je nezbytné nejprve pochopit pojmy systém a informace jako takové.

Pojem informace je součástí pojmového aparátu každého člověka. Významový obsah tohoto pojmu je značně široký, a proto lze informaci chápat i obecně ve smyslu sdělování nějaké zprávy, poznatku, události či jevu. Z filozofického hlediska je informace nehmotná a představuje to, co je vnímáno člověkem. Kvalitní informace snižuje nejistotu a riziko. Musí však být cílená, včasná, přesná, musí jí být přiměřené množství a musí být srozumitelná (prezentována vhodnou formou). Kvalitu informace ovlivňuje v nemalé míře také její cesta od zdroje k příjemci, na které může být jak záměrně, tak neúmyslně zasažena chybami či zmanipulována, a to až do té míry, že se stává dezinformací. [9]

Vycházíme-li z teorie informace, pak je informace zpráva, která nám upřesňuje určitá fakta o jevech nebo objektech reálného světa a vyjadřuje se v bitech (BIT – je zkratka slov „binary digit“ a představuje číslici, která nabývá hodnot 0 a 1). V současnosti jsou informace vnímány jako výrobní zdroj. Je proto potřebné informace efektivně získávat a využívat je. [9]

Samotný pojem systém se používá v mnoha souvislostech a jeho význam závisí na historickém vývoji poznatků. Je blízký pojmům celistvost, organizace, organizmus, struktura. Původně ve starořecké filozofii znamenal seskupení, sjednocení, celek. Později se objevila myšlenka o řádu a uspořádanosti prvků nebo části systému. Představa o struktuře vznikla již v antickém myšlení a uplatnila se zejména v tehdejších poznatcích o stavbě živého organizmu. [9]

Dnes je **systém** chápán **jako účelově definovaná množina prvků a vazeb mezi nimi** a pojem systém se užívá jako označení určité části reálného světa s charakteristickými vlastnostmi. Systémy můžeme rozdělit na systémy přirozené a umělé. Systémy přirozené jsou takové systémy, jejichž hlavní části nejsou a nebyly vytvořeny člověkem a existují nezávisle na něm. Pokud hovoříme o systémech umělých, hovoříme o systémech vytvořených člověkem. Informační systém je tedy systémem umělým a člověk může výrazně ovlivňovat jeho kvalitu. [9]

Informační systém lze definovat jako soubor lidí, metod a technických prostředků zajišťujících sběr, přenos, uchování, zpracování a prezentaci dat s cílem tvorby a

poskytování informací dle potřeb příjemců informací činných v systému řízení. Tato definice zahrnuje člověka jako součást informačního systému a zmiňuje se o míře potřeby příjemců informací. Informační systém je účelnou formou využití informačních technologií v sociálně-ekonomických systémech. Informační systém se skládá z následujících komponent:

- **technické prostředky (hardware)** – počítačové systémy různého druhu a velikosti, doplněné o potřebné periferní jednotky, které jsou v případě potřeby propojeny prostřednictvím počítačové sítě a napojeny na paměťový subsystém pro práci s velkými objemy dat,
- **programové prostředky (software)** – tvořené systémovými programy, řídícími chod počítače, efektivní práci s daty a komunikaci počítačového systému s reálným světem, a programy aplikačními, řešícími určité třídy úloh určitých tříd uživatelů,
- **organizační prostředky (orgware)** – tvořené souborem nařízení a pravidel, definujících provozování a využívání informačního systému a informačních technologií,
- **lidská složka (peopleware)** – řešení otázky adaptace a účinného fungování člověka v počítačovém prostředí, do kterého je vřazen,
- **reálný svět (informační zdroje, legislativa, normy)** – kontext informačního systému.

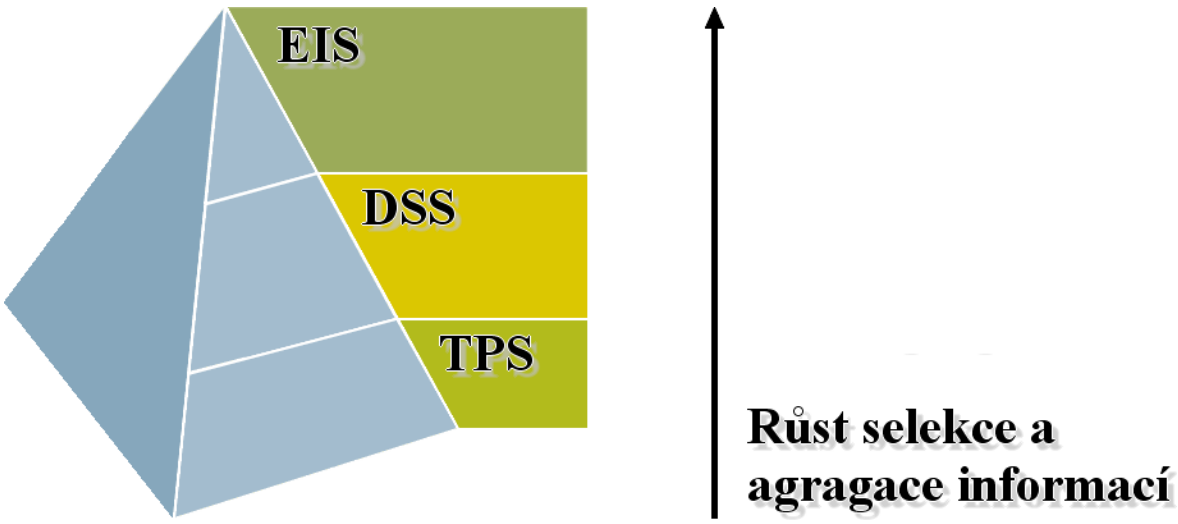
Má-li být informační systém firmy či instituce efektivní, nesmí být při jeho vývoji zanedbána žádná z jeho složek. [9]

Podle úrovně řízení a růstu neurčitosti lze informační systémy rozdělit na systémy transakční (TPS, Transaction Processing Systems), systémy pro podporu rozhodování (DSS, Decision Support Systems) a systémy pro podporu vrcholového vedení (EIS, Executive Information Systems).

Transakční systémy (TPS) tvoří základ informační pyramidy (viz následující strana obr. 2-1). TPS jsou následovníky klasických dávkových systémů pro mechanizaci agendových úloh, jako jsou mzdy, fakturace, inventarizace apod. Pracují na operativní úrovni a dalo by se říci, že zajišťují základní procesy v organizaci. [9]

Systémy pro podporu rozhodování (DSS) mají schopnost provádět rozmanité analýzy dat bez potřeby složitého ovládání. Jsou především určeny pro podporu středních složek managementu (poskytují mu komfort). Jedná se o počítačovou podporu metod rozhodovací analýzy a operační systémové analýzy. Tyto systémy předpokládají, že uživatel rozumí podstatě metody a ví, jaká vstupní data do systému zadat. Výsledkem úspěšné kooperace uživatele se systémem je „rychlé a snadné“ zobrazení požadovaných výsledků. [9]

Manažerské aplikace byznys inteligence (EIS) zabezpečují vrchol řídící pyramidy. Slouží především vrcholovému vedení organizace, které se více zajímá o informace z okolí organizace (technické inovace, trh, legislativa, konkurence apod.). Jsou navrhovány tak, aby měly přístup k externím datům a zároveň byly napojeny na ostatní části informačního systému firmy. Data obsažená v EIS mají obvykle vysokou vypovídací hodnotu. Tyto systémy jsou rovněž charakterizovány intuitivním ovládáním. [9]



Obrázek 2-1: Informační pyramida

Zakázkový systém firmy Telko s r.o., kterému se budu v dalších částech diplomové práce věnovat, je z tohoto hlediska systémem transakčním (TPS). Je zaměřen na sběr a zpracování informací o zakázkách a zajišťuje tak některé základní procesy v organizaci. V budoucnu by se tento systém mohl stát jedním ze základních pilířů komplexního informačního systému firmy.

2.2.2 Software z pohledu měření kvality

Na první pohled by se mohlo zdát, že software je výrobek jako každý jiný. Druhý pohled však odhalí, že je tato představa poněkud zkreslená. Software se odlišuje způsobem „výroby“, částečně také způsobem vývoje i používání a také procesem vývoje. Proč se tedy software liší od jiných výrobků? Hlavním rozdílem je pojetí kvality očima zákazníka/zadavatele. Kvalita každého výrobku je charakterizována podle jeho vlastností. Tyto vlastnosti bývají zpravidla viditelné (vnější) nebo neviditelné (vnitřní). [6]

Představme si takovou ledničku. Je to plechový kvádr vážící několik (desítek) kilogramů. Tento kvádr má na první pohled viditelné charakteristiky (velikost, barvu, konkrétní tvar, ovládací prvky, několik poliček a přihrádek pro různé potraviny). Všechny tyto atributy jsou exaktně změřitelné, popsatelné, specifikovatelné, porovnatelné. Bude-li spotřebitel uvažovat o koupi ledničky, může se na základě těchto atributů snadno rozhodnout. Projeví-li se na ledničce po její koupi nějaká závada, půjde ji vrátit zpět do obchodu. Všechny výše uvedené viditelné vlastnosti samozřejmě exaktně ledničku nespecifikují, avšak jistě mi dáte za pravdu, že by se na jejich základě koupě dala realizovat. Software lze pomocí viditelných vlastností rovněž charakterizovat. V tomto případě hovoříme o charakteristice uživatelského rozhraní (Jak vypadají dialogy? Jak přívětivě program komunikuje? Jakou obsahuje nápovědu? Jak řeší nečekané problémy a chybové stavy?), neboť právě uživatelské rozhraní je tím jediným, s čím uživatel přichází do kontaktu. Tyto údaje však ve většině případů ani zdaleka neodpovídají výsledkům snažení vývojářského týmu. Pokud jsem napsal, že ledničku nelze exaktně charakterizovat pomocí jejích viditelných vlastností, u softwaru toto platí dvojnásob (ne-li více). [6]

Lednička i software mají druhou skupinu vlastností, nazvěme je „neviditelné“. V případě ledničky se jedná např. o výkon, spotřebu, použité komponenty, použitou chladící kapalinu, životnost apod. Také pro software existuje řada speciálních metrik, které dokáží změřit „neviditelné“ vlastnosti softwaru. Kromě toho existují sofistikované nástroje (např. Borland Together Control Center), které dokáží požadovaný software kompletně změřit, popsat a dokonce i ohodnotit a klasifikovat. Nejjednodušší softwarová metrika je označována LOC a znamená Lines of Code (počet řádků programového kódu). Toto hledisko měření kvality softwaru může být však značně zavádějící. [6]

Kromě základní metriky LOC existují i jiné. Metriky lze dělit do skupin podle jejich typu a také podle toho, co vlastně měří. Ve skupině objektově orientovaných metrik nalézáme počet tříd, počet metod na třídu, počet atributů na třídu, počet modulů, maximální hloubku v hierarchii tříd, počet podtříd, počet vstupů (globálních proměnných, parametrů), soudržnost apod. V další skupině se objevuje počet zdrojových souborů, počet funkcí/procedur, počet příkazů, počet prázdných řádků, průměrný počet řádků na funkci apod. Měřit lze i způsob komunikace mezi objekty, ale nakonec i kvalitu, čitelnost, eleganci a dokumentaci zdrojového kódu apod. [6]

Další kategorie metrik (hovoříme o modelu FURPS+) se zabývá funkcionalitou, použitelností, spolehlivostí, výkonem a možnostmi podpory, tj. vlastně vnějšími projevy kvality a také celkovými náklady na provozování (total cost of ownership). Mezi metriky kategorie FURPS+ patří např.:

- **funkcionalita** – vlastnosti, schopnosti, bezpečnost,
- **použitelnost** – vztah k lidskému faktoru, estetika, vnější konzistence, dokumentace,
- **spolehlivost** – frekvence chyb, zotavitelnost po chybách, přesnost, střední doba k výpadku,
- **výkon** – rychlost, efektivita, spotřeba zdrojů, výkon, propustnost, doba odezvy,
- **podpora** – testovatelnost, rozšiřitelnost, přizpůsobitelnost, obtížnost údržby, konfigurovatelnost, lokalizace apod.

Nevýhodou softwarových metrik je, že jsou pro zákazníka/zadavatele většinou zcela nesrozumitelné a nedají se pro tento účel použít. Tyto metodiky reprezentují pohled zevnitř, který zákazníka v podstatě nezajímá. Metriky jsou tedy vhodné zejména pro interní ověřování kvality v rámci vývojového týmu. [6]

Nejdůležitějším aspektem měření kvality softwaru je vždy spokojený zákazník. I zdánlivě špatný, zbytečný a nesmyslný program může být pozitivně ohodnocen a denně využíván. A přesto by ve srovnání s jinými programy téže funkce pohořel. To stejné platí i v opačném případě. Kvalitu software je vždy možné měřit jen ve vztahu k očekáváním, která na něj klade jeho uživatel. [6]

2.2.3 Softwarové inženýrství

Softwarové inženýrství je ve srovnání s ostatními inženýrskými obory velmi mladým oborem (přelom 60. a 70. let 20. století) a zjednodušeně řečeno se zabývá efektivní tvorbou software. Je to obor, který vnáší řád do světa softwarových vývojářů. Existuje celá řada definic, upřesňujících toto slovní spojení. Pro účely diplomové práce jsem zvolil definici, kterou pronesl Fritz Bauer, jeden z duchovních otců softwarového inženýrství. Tato definice poprvé zazněla na konferenci NATO v roce 1968, kde se pojem „softwarové inženýrství" živě diskutoval. Definice zní takto: **„Softwarové inženýrství je zavedení a používání řádných inženýrských principů tak, abychom dosáhli ekonomické tvorby softwaru, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích."** [6]

Pokud bychom shrnuli tuto definici, tak bychom zjistili, že úkolem programátora (softwarové firmy) není jen „napsat program". Jeho počínání by mělo zahrnovat celou řadu dalších činností. Souhrn těchto činností je specifikován právě softwarovým inženýrstvím jako takovým.

V definici se objevuje slovní spojení **ekonomická tvorba softwaru**, tedy „po všech stránkách úspěšnou" tvorbu softwaru. Můžeme nalézt řadu faktorů vedoucí k naplnění tohoto cíle. Příklady těchto faktorů uvádím v následujícím přehledu.

- **Vhodně sestavený vývojový tým** nesmí mít ani málo ani mnoho členů, měl by obsahovat vhodné osobnostní typy, měl by pokrývat všechny potřebné role, měl by respektovat používanou metodiku vývoje, měl by mít vhodný truck faktor (závislost celého týmu na znalostech jeho jednotlivých členů, aby se nestalo, že po odchodu klíčového člena nerozumí dané problematice už nikdo jiný a výsledkem je totální krach projektu), měl by být zárukou efektivní práce lidí (je jedním ze základních pilířů ekonomické tvorby softwaru).
- **Volba správného vývojového nástroje/operačního systému** bývá většinou součástí zadání projektu, někdy však lze na toto téma diskutovat a firma může zákazníkovi navrhnout vhodnou alternativu, která bude zároveň nejvíce vyhovovat znalostem a vybavením vývojářského týmu.

- **Úvaha vyvinout/koupit**. Většinou se nevyplácí dělat celý vývoj vlastními silami, zvláště v případě systémů, u kterých lze předpokládat, že už je někdy někdo dělal. V takovém případě se můžeme poohlédnout třeba po existujících knihovnách: pokud budeme při hledání úspěšní, může zakoupení knihovny (včetně nákladů na její prozkoumání a interakci) uspořit nečekané procento nákladů.
- **Nalezení společné řeči se zadavatelem**. Úspora, která se promítne v budoucnu. Pokud selžeme v tomto aspektu, povede to k tomu, že buď v úvodní fázi zakázku vůbec nezískáme, dojde ke sporům při vývoji nebo k odmítnutí převzetí díla (v závěrečné fázi).
- **Úvaha o budoucí údržbě/rozšiřování programu**. Pokud dnes někde při vývoji ušetříme pár tisíc, protože se nám zdá být dražší řešení zbytečně robustní, může nás později stát požadavek na úpravu měsíc času. Vždy záleží na konkrétní situaci a rozhodovat by v tomto případě měl zkušený pracovník týmu. Podstatné je, že správná architektura je dalším pilířem ekonomické tvorby softwaru.

Takových bodů bychom nalezli celou řadu a všechny můžeme zahrnout do pojmu „úspěšná a ekonomická tvorba softwaru“, protože zanedbání kteréhokoliv z nich povede k ekonomickým (i jiným) ztrátám – buď ihned nebo v budoucnu. [6]

Jak jsem se již zmínil, historie softwarového inženýrství je poměrně krátká. Etapu do poloviny 60. let 20. století můžeme označit jako průkopnická léta. Obvyklí programátoři této doby jsou zaprvé šťastlivci, kteří měli přístup k (tehdy velmi drahému) hardwaru, a za druhé nadšenci, kteří vytvářejí zpravidla jednoúčelové programy ušité „na míru“ danému počítači a dané architektuře. O nějakém komplexním, řízeném přístupu k vývoji nemůže být řeč. Vytvořené programy jsou zpravidla neudržovatelné a neměnné. Na přelomu 60. a 70. let se začínají objevovat první náznaky snah, později ústících ve vznik softwarového inženýrství. Téma „softwarové inženýrství“ se dokonce objevuje na konferenci NATO v roce 1969. Vznikají pojmy „návrh shora dolů“, „modularita“ atd. Začíná být chápán význam správně složeného týmu. V 70. letech už softwarové inženýrství vzniká jako obor. Tvorba programů začíná být častějším jevem, hardware se stává dostupnějším. Začínají se vytvářet první aplikace umožňující interakci se svými uživateli. Vývoj softwaru však není nijak řízen a neexistují žádné zavedené postupy, což vede

k problémům s dodávkami a nedokončeným projektům. V letech 1976-1977 se však již začínají využívat mnohé dnes známé techniky softwarového inženýrství, např. specifikace, návrh, architektura, testování, zajištění kvality, modely životního cyklu apod. V 80. letech, zároveň s rozvojem softwaru jako takového, se masivně rozvíjí i softwarové inženýrství. Dochází k rozmachu objektově orientovaných (OO) přístupů. Teprve v roce 1997 je však softwarové inženýrství uznáno jako obor s certifikátem v USA. [6]

Co však ve skutečnosti předcházelo vzniku softwarové inženýrství? Bylo to v době končících 60. let minulého století, kdy světem softwaru otřáslo obtížné období, později označené jako **softwarová krize**. Hlavními charakteristikami softwarové krize bylo neúnosné prodražování a prodlužování projektů, nízká kvalita programů, nesnadnost/nemožnost údržby a inovací, špatná produktivita práce programátorů, neefektivita vývoje, nejistota výsledku (do posledního okamžiku bylo nejasné nejen to, zda se projekt vyplatí, ale dokonce i to, zda se vůbec dokončí) a řada dalších negativních znaků. Mezi důvody, proč došlo před nástupem softwarového inženýrství k softwarové krizi, můžeme jistě zařadit problémy uvedené v následujících odrážkách.

- **Špatná komunikace** na všech úrovních: zákazník/analytik (dříve funkce analytika a programátora splývala, což mohlo vést k vzájemnému nepochopení), analytik/vývojář, vývojář/jiný vývojář, vývojář/šéf.

  Dnes je funkce analytika ve většině případů ceněna více než funkce vývojáře (kodéra) a na samotnou komunikaci a vztahy mezi členy týmu se zaměřuje stále větší pozornost.

- **Nesprávný přístup** programátorů byl dříve rovněž problémem. Hlavním cílem programátorů bylo často „vyřádit se“, předvést, seberealizovat. Podle této konvence byl většinou uspokojen programátor, nikoliv zákazník. Tento přístup vedl k přesvědčení, že pravdu má vývojář, neboť on jediný ví, „jak to jde naprogramovat“.

  V dnešní době jsou používány např. systémy CRM (Customer Relationship Management, systém řízení vztahů se zákazníky) a manažeři si stále více uvědomují nutnost orientace na zákazníka a jejich požadavky. Teprve podle požadavků musí tým zvolit vhodnou metodiku budoucího vývoje. Pro vývojáře je nutné pracovat pro tým a ne pro vlastní seberealizaci.

- **Nesprávné odhady** času, ceny, rozsahu, efektivity apod. způsobily krach již řady projektů. Dříve se kladl hlavní důraz na psaní kódů než na samotný návrh a důkladnou analýzu.

  Dnes například víme, že firmy, které skutečně dbají na kvalitu svých výstupů, mají průměrný nárůst 30-50 řádek programového kódu na programátora a den. Přesto však vytvoření optimálního odhadu představuje složitý problém, protože jej děláme na úplném začátku projektu, kdy je pro jeho vytvoření nejméně informací.

- **Špatné plánování**. Bylo obtížné vypracovat plán, který by byl přijatelný pro zákazníka a splnitelný pro vývojáře (ostatně je to velmi obtížné i dnes). Neexistovaly empiricky ověřené příklady, ze kterých by se vyšlo. Vývoj probíhal heuristicky.

  Dnes jsou k dispozici propracované metody pro plánování jednotlivých částí projektu, např. pro plánování nákladů, časových plánů, zdrojů apod. Subjekty zodpovědné za plánování mají v současnosti celou řadu empiricky ověřených příkladů. Metodiky vývoje softwaru na proces plánování dbají, a tak často předepisují provádět průběžné, iterativní plánování společně s opakovaným posuzováním a ověřováním vztahu realita vs. plán.

- **Nízká produktivita práce** souvisela s tím, že se programátoři často nezabývali tím, co bylo nejvíce potřeba. Programátoři vytvářeli nezávislé fragmenty kódu a při jejich integraci mohlo docházet k problémům.

  Dnes je v týmu zastoupena řada rolí. Každý člen by měl provádět to, v čem je nejsilnější a pro tým nejpřínosnější. Klíčem k produktivitě práce je souhra jednotlivých rolí a správné přiřazení priorit požadovaným úlohám.

Dále bych mohl pokračovat problémy jako je **podcenění hrozeb a rizik**, **neznalost základních pravidel** (metod, postupů), **nezvládnutí technologie** atd. Softwarová krize měla řadu příčin. Pro svět softwarového inženýrství je však důležité, že se postupně začaly formovat více či méně inženýrské postupy, jejichž dodržování slibuje odstranění specifických problémů období softwarové krize. Tyto postupy, pokrývající postupně všechny fáze vývoje softwaru, souhrnně tvoří disciplínu softwarového inženýrství, jak ji známe dnes. [6]

2.2.4 Metodiky, životní cykly a procesy vývoje softwaru

V této části se podíváme na to, jakým způsobem vznikaly, zanikaly a rozvíjely se metodiky vývoje softwaru. Než začnu s výkladem, udělám nejprve jasno v používaných pojmech a jejich významu.

V oblasti softwarového inženýrství se vyskytují zejména tři pojmy, které často různí lidé různě zaměňují a může tak snadno docházet k zásadním problémům. Těmi pojmy jsou metodologie, metodika a metoda. Nyní se pokusím tyto pojmy blíže specifikovat.

- **Metodologie** je nejobecnější pojem a znamená ve své podstatě „nauku o metodikách“. Jinými slovy se pod tímto pojmem skrývá vědní disciplína, která nějakým způsobem rozebírá metodiky, definuje je apod.
- **Metodika** je v oblasti softwarového inženýrství označení pro komplexní postupy a návody pro vývoj softwarové aplikace. Pod metodikou se skrývají všechny etapy řešení (u vývoje softwarové aplikace jde tedy o všechny fáze životního cyklu). Metodika se zabývá spíše pohledem z výšky a hledá odpovědi na otázky typu **proč**, **kdo**, **kdy** a **co**. Metodika nemusí nutně rozebírat způsoby, jak nebo pomocí čeho danou operaci provést. Příkladem metodiky je například WebWAVE Development Process, který pokrývá všechny fáze vývoje internetové aplikace.
- **Metoda** je pak označení pro nějaký konkrétní postup vedoucí k vyřešení dílčího problému. Příkladem metody může být např. Yourdonova strukturovaná metoda (Yourdon Structured Method, YSM), která definuje posloupnost kroků směřujících k realizaci strukturální analýzy systému.

Metodiky vývoje aplikací jsou jedněmi z nejdůležitějších produktů softwarového inženýrství. Stejně jako má svou historii tento inženýrský obor, procházely jistým vývojem i metodiky. Při tom platí, že se metodiky vždy snažily odrážet aktuální situaci v době svého zvyku. To znamená, že se pokoušely přizpůsobit vývoj softwaru konkrétním požadavkům kladeným na software v dané době. Cílem metodik bylo odstranit nedostatky vývoje aplikací, které se v příslušném období nejvíce projevovaly. [6]

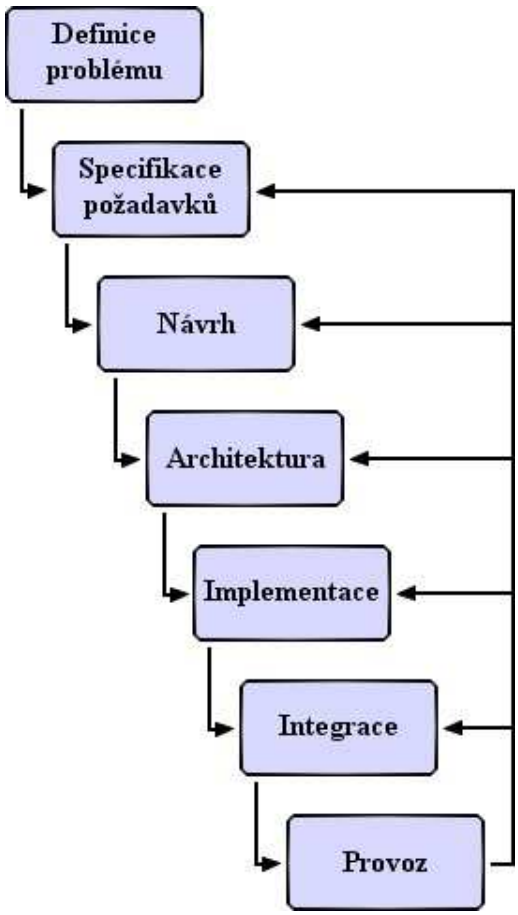
Pojďme se nyní postupně podívat na jednotlivé metodiky tak, jak v průběhu doby vznikaly:

- **Model napiš a oprav** (Code and Fix Model, příp. Build and Fix Model) se používal od úplných prvopočátků vývoje programů (v 50. letech minulého století). Tento model popisuje obrázek 2-2. V podstatě se jedná o sepsání aplikace, předání do provozu (spuštění) a v opravování chyb. V případě tohoto postupu nelze snad ani hovořit o metodice, model vznikl zcela spontánně a i jeho pojmenování spatřilo světlo světa až mnohem později. O efektivitě tohoto modelu a o aplikacích, které na jeho základě vznikly, nelze říct moc pozitiv.

Obrázek 2-2: Schéma modelu napiš a oprav (Code and Fix Model)

- **Striktní posloupnost fází** (Stagewise Model), definovaný již v roce 1957, rozděluje vývoj softwaru do několika částí – definice problému, specifikace požadavků, architektura a návrh, implementace, integrace, provoz. Zásadním problémem tohoto modelu je naprostá absence zpětné vazby. Podle tohoto modelu se po skončení fáze neprováděly žádné revize, nehodnotily se dosavadní výsledky, nerevidovaly se požadavky a nehledala se žádná rizika. Vývoj postupoval pouze kupředu. Návrat do předchozí fáze např. za účelem bližší specifikace požadavků byl nemyslitelný. Jediné navrácení bylo možné až v samotném závěru – po dodání aplikace mohla následovat fáze tzv. revalidace.
- **Vodopádový model** (The Waterfall Model) vzniká v roce 1970 a je výsledkem nově zformovaného oboru softwarového inženýrství. Vodopádový model se od svého předchůdce (Stagewise Model) liší především snahou o zpětnou vazbu. Na konci fáze dojde k jejímu

vyhodnocení a jejímu případnému přepracování/opravení. Vodopádový model se v podstatě skládá ze stejných fází jako model stagewise, avšak v jeho schématu (viz obrázek 2-3) jsou patrné i šipky vedoucí zpět mezi jednotlivými fázemi.

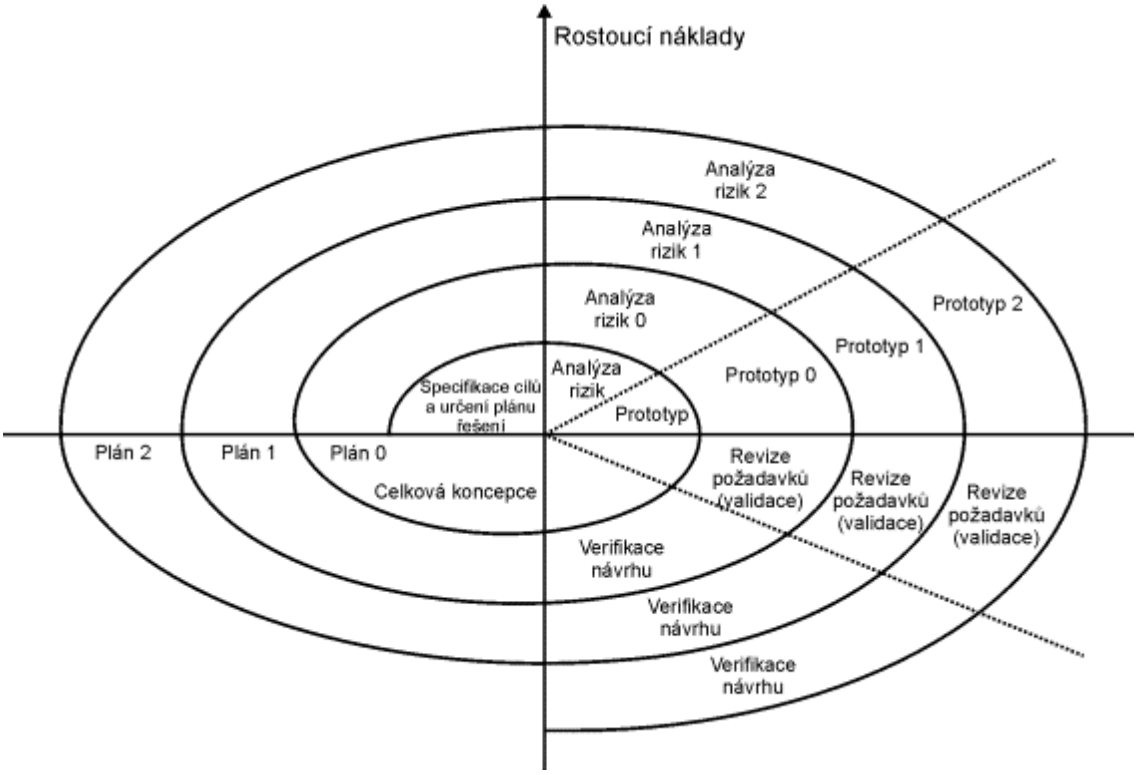


Obrázek 2-3: Posloupnost fází v modelu Stagewise

Vodopádový model znamenal ve své době malou revoluci a mnoho autorů jej považuje za skvělý výchozí bod pro vývoj softwaru. Mezi jeho výhody patří jasné oddělení jednotlivých fází (umožňující dobře kontrolovat dosažený pokrok), možnost vrátit se při vývoji o krok zpět a také jeho jednoduchost. Přesto však má tento model i nevýhody a to například dodání formou „velkého třesku“ (zákazník dlouho nevidí žádnou aktivitu a nakonec dostane kompletní, hotový produkt), jistá nepružnost (způsobující dlouhé

dodací lhůty) apod. Vodopádový model se dále rozvíjí a různí autoři jej upravují a vylepšují.

- **Spirálový model** byl navržen Barry Boehmem v roce 1985. Spirálový model reaguje na některé nedostatky vodopádového modelu, které postupně začaly být více či méně patrné. Do procesu vývoje jsou zavedeny dva přelomové koncepty – iterativní přístup a opakovaná, důsledná analýza rizik. Vzhledem k tomu, že je na začátku projektu obtížné dopodrobna specifikovat veškeré požadavky, stanoví se v úvodu pouze obecný, vnější rámec. Následně se vyvine část aplikace, provede konzultace se zákazníkem a podle konkrétní situace se pokračuje analogicky dalším krokem. Vývoj tedy probíhá v opakovaných krocích – iteracích. Jednotlivé kroky se opakují a projekt tak postupně nabývá na rozsahu (hovoříme o spirálách, viz obrázek 2-4). Důvodem tohoto postupu je dostat výsledný produkt do takové podoby, aby co nejvíce odpovídal požadavkům zákazníka.



Obrázek 2-4: Schéma spirálového modelu životního cyklu

Kromě modelů uvedených v předchozích odstavcích však vzniká i řada dalších přístupů. Pokud bych měl vyjmenovat následníky vodopádového modelu, zmínil bych se o modelech tzv. **Test Development** (vodopádový model rozšířený o specifikaci a provádění testů na konci každé fáze), **Exploratory Programming** (první předchůdce iterativního přístupu – zpočátku se stanoví pouze základní požadavky a vyvine se hrubá kostra systému, která se následně rozšiřuje podle požadavků zákazníků), **Prototyping Model** (vodopádový model rozšířený o vytvoření prototypu za účelem zpřesnění požadavků), **Evolutionary/Incremental Model** (další předchůdce iterativního programování), **Transformation Model** (implementace je vlastně transformací formální specifikace za použití jasně stanovených pravidel) apod. [6]

V 70. letech se rozvíjejí strukturované metodiky a dílčí metody řešící jednotlivé fáze vývojového cyklu (např. **Yourdonova**, **Hatleyova**). 80. léta zaznamenávají vznik některých komplexnějších metodik pokrývajících celý proces vývoje aplikací (např. **SSADM**). V 90. letech spatřují světlo světa první zástupci objektově orientovaných metodik (např. **Rational Unified Process**). Začátkem 3. tisíciletí pak vznikají tzv. **agilní metodiky** specifické svou nižší mírou formálnosti. Mezi agilní metodiky řadíme např. **extrémní programování**, metodiku **Crystal**, **SCRUM Development Process**, **Aspect Oriented Programming**, **Test Driven Development** apod. [6]

Jak lze vyčíst z této podkapitoly, metodik pro tvorbu softwaru existuje celá řada. Každá má své silné a slabé stránky. Volba správné metodiky je závislá především na typu a rozsahu projektu. Každý projekt je unikátní, proto neexistuje žádná univerzální metodika, která by za nás vše vyřešila.

Pro vývoj zakázkového systému firmy Telko s r.o. použiji inkrementální přístup s krátkými iteračními cykly (14-30 dnů). V první fázi vývoje sestavím úvodní studii, ve které popíši, jak současný systém funguje. Poté představím svou vizi. V další fázi provedu specifikaci uživatelských požadavků a provedu hrubý návrh aplikace. Dál přijde na řadu několik iterací (návrh, implementace, testování). Na konci každé iterace vznikne nová verze aplikace (doplněná o novou funkcionalitu), která bude předložena zákazníkovi k otestování. Pokud bude mít zákazník nějaké připomínky, promítnou se do další iterace. Vývojový cyklus skončí v okamžiku, když funkcionalita programu bude z pohledu zákazníka dostačující.

2.2.5 UML – Úvod

Sjednocený modelovací jazyk (UML, Unified Modeling Language), je druh grafické notace podporovaný nezávislým meta-modelem, který umožňuje popisovat a navrhovat softwarové systémy, konkrétně systémy budované využitím objektově orientované (OO) metodiky. To je poněkud zjednodušená definice. Ve skutečnosti si různí lidé pod pojmem UML představují několik různých věcí. To je způsobeno jednak vlastní historií UML a jednak různými názory na to, co to je „efektivní proces softwarového inženýrství“. Grafické modelovací jazyky jsou v softwarovém průmyslu přítomny již mnoho let. Jejich základním hnacím motorem je to, že programovací jazyky nejsou na tak vysoké úrovni abstrakce, aby napomáhaly při diskusích o návrhu. [3]

UML je relativně volným standardem řízeným uskupením Object Management Group (OMG), jež je volným konsorciem společností. OMG vzniklo za účelem vytváření standardů, které by podporovalo vzájemnou slučitelnost, konkrétně slučitelnost objektově orientovaných systémů. Nejznámějším produktem OMG je nejspíše CORBA (Common Object Request Broker Architecture). UML vzniklo sjednocením mnoha objektově orientovaných grafických modelovacích jazyků, které vznikaly na konci 80. a počátkem 90. let 20. století. [3]

2.2.6 UML – Způsob použití

Obecně můžeme říci, že existují 3 způsoby využití UML. Prvním z nich je UML pro tvorbu náčrtků (**UML as sketch**). Náčrtky neslouží pro detailní popis sytému. Jejich síla je zejména v usnadnění komunikace mezi vývojáři. Tyto náčrtky lze využít u **dopředného inženýrství** (forward engineering) i u **zpětného inženýrství** (reverse engineering). Pokud hovoříme o dopředném inženýrství, znamená to, že z patřičných UML diagramů vytváříme programový kód. U zpětného inženýrství je tomu naopak. Z programového kódu se vytvářejí UML diagramy. Zpětné inženýrství má význam tehdy, pokud potřebujeme zjistit a popsat logiku již existujícího programu a to nastává zejména v okamžiku, kdy k danému programu neexistuje žádná dokumentace. [3]

Dalším způsobem využití je UML pro podrobný návrh (**UML as blueprint**). Jeho hlavní vlastností je kompletnost. V dopředném inženýrství se předpokládá, že návrhář připraví pro programátory detailní návrh, který vezmou a zakódují. Programátor by tedy

měl minimalizovat své myšlení. Návrh může obsahovat všechny detaily, muže se však také soustředit pouze na určitou oblast. Návrhář většinou vytvoří modely rozhraní subsystémů a samotnou implementaci detailů ponechá na programátorech. Proces tvorby detailních návrhů je komplikovaný, proto se pro usnadnění práce vývojářů využívají speciální CASE (computer-aided software engineering) nástroje. Hlavním rozdílem mezi náčrtkem a detailním návrhem spočívá v podrobnosti návrhu. Hranice bývá často neostrá. [3]

Pokud má vývojářský tým k dispozici obzvlášť výkonné CASE nástroje, lze **UML** využít i **jako programovací jazyk**. Tato varianta využití UML je poněkud extrémní a v praxi se příliš nevyužívá, avšak dokáže uspořit značné množství času při implementaci návrhu. Díky těmto nástrojům se obvykle vygeneruje základní kostra programu a případné nevyřešené detaily se nechají programátorům. Výhodou tohoto řešení je, že detailně provedená specifikace softwaru pomocí UML lze poměrně jednoduše přenášet na různé platformy. V tomto pojetí UML je nemožné rozlišovat mezi dopředným a zpětným inženýrstvím, protože návrh software představuje vlastním programový kód. [3]

V dalším textu diplomové práce budu využívat UML formou náčrtku (UML as sketch) a tyto náčrtky budu využívat zejména tehdy, pokud povedou k lepšímu pochopení určitých problematických oblastí v rámci zakázkového systému firmy Telko s r.o.

2.2.7 UML – Diagramy

UML verze 2 popisuje celkem 13 typů oficiálních diagramů. Jejich výčet a stručná charakteristika je uvedena v tabulce 2.1 na další straně. Diagramy UML lze rozdělit do dvou typů (viz příloha P1) a to na diagramy struktur (Structure Diagram) a diagramy chování (Behavior Diagram). Ve své práci všechny tyto diagramy zřejmě nevyužiji, ale troufám si již nyní tvrdit, že přinejmenším využiji diagram tříd, diagram sekvenční, diagram případů užití a diagram aktivit. Ukažme si teď alespoň stručný popis těchto čtyř diagramů.

Diagram tříd (class diagram) je nejen široce užívaný, ale je též začleněn ve velké řadě modelovacích konceptů. Diagram tříd tedy popisuje typy objektů v systému a různé druhy statických vztahů, které mezi nimi existují. Diagramy tříd ukazují i vlastnosti a operace tříd, a také omezení týkající se způsobu, jakým jsou objekty spojovány. Diagramy tříd tvoří páteř celého UML. [3]

Sekvenční diagram (sequence diagram) typicky zachycuje chování jednoho scénáře. Ukazuje několik vzorových objektů a zpráv, které jsou předávány mezi těmito objekty v rámci daného případu užití. Sekvenční diagramy vynikají v zobrazování spolupráce mezi objekty, ale už tak nevynikají v precizní definici chování (vhodnější je použít diagram stavový – zachycuje chování objektu napříč různými případy užití). [3]

Diagram případů užití (use case diagram) je metodou pro zachycení funkčních požadavků na systém. Případy užití popisují typické interakce mezi uživateli systému a samotným systémem, a předkládají nám příběh o tom, jak je systém používaný. Případ užití je sada scénářů, které jsou dohromady svázané společným cílem uživatele. [3]

Diagram aktivit (activity diagram) je technikou určenou k popisu procedurální logiky, business procesů a toku práce (work flow). V mnoha ohledech hrají podobnou roli, jako vývojové diagramy (flowcharts), ale hlavní rozdíl mezi těmito dvěma notacemi je v tom, že diagramy aktivit jsou schopny zachytit paralelní chování. [3]

Diagram	Účel	Původ
Aktivit	Procesní a paralelní chování	V UML 1
Balíčků	Hierarchická struktura kódu pro překlad	Neoficiálně v UML 1
Časování	Interakce mezi objekty; důraz na časování	Nový v UML 2
Komponent	Struktura a propojení komponent	V UML 1
Komunikace	Interakce mezi objekty; důraz na spojení	V UML 1 diagram spolupráce
Nasazení	Nasazení artefaktů na uzly	V UML 1
Objektů	Příklad uspořádání instancí	Neoficiálně v UML 1
Přehledu interakcí	Kombinace sekvenčního diagramu a diagramu aktivit	Nový v UML 2
Případů užití	Jak uživatelé komunikují se systémem	V UML 1
Sekvenční	Interakce mezi objekty; důraz na sekvence	V UML 1
Složených struktur	Dekompozice tříd za běhu programu	Nový v UML 2
Stavový	Jak události mění objekt během jeho života	V UML 1
Tříd	Třídy, vlastnosti a vztahy	V UML 1

Tabulka 2.1: Oficiální typy diagramů v UML [3]

2.2.8 ER modelování

Entitně relační model (Entity-relationship model, ERM) se v softwarovém inženýrství používá pro abstraktní a konceptuální znázornění dat. ER modelování je metoda datového modelování, která vytváří jeden z typů konceptuálních schémat či sémantických datových modelů systému (obvykle relační databáze) a požadavků na něj stylem shora dolů. Diagramy vytvořené pomocí této metody se nazývají entity-relationship diagramy, ER diagramy nebo také zkráceně pouze ERD. Konečnou podobu dostalo ER modelování v práci Petera Chena z roku 1976. [1i]

Techniky datového modelování se používají pro popis ontologie (tj. přehled a klasifikace použitých pojmů a jejich vztahy mezi sebou) pro určitou oblast zájmu. V případě, že jde o návrh informačního systému založeného na databázi, je konceptuální model v pozdější fázi (obvykle nazývané logický návrh) namapován na logický datový model, kterým je kupříkladu relační model, ten je dále mapován ve fyzické fázi na fyzický datový model. Je důležité poznamenat, že obě tyto fáze se obvykle berou za "fyzický návrh". Pateří ER modelu jsou tři prvky – entity, relace a atributy.

- **Entita** (entity) se dá definovat jako věc schopná samostatné existence a je jednoznačně identifikovatelná. Hovoříme-li o entitě obvyklým způsobem, hovoříme určitém aspektu reálného světa, který se dá od ostatních aspektů odlišit. Entita může být fyzicky existující objekt, jako např. dům nebo automobil, nebo událost jako je prodej domu nebo servis automobilu nebo může jít o pojem jako je např. zákaznická transakce nebo objednávka. Přesto, že termín entita se užívá nejčastěji, bylo by podle Chena dobré opravdu odlišovat mezi entitou a entitním typem. Entitní typ je kategorie a entita v pravém slova smyslu je instance daného entitního typu, a toho obvykle existuje mnoho instancí. Protože je entitní typ poněkud těžkopádný výraz, lidé obvykle raději používají výraz entita jako synonymum. Entity jsou označovány podstatnými jmény (např. počítač, zaměstnanec, píseň, matematická teorie). Entity se zobrazují jako obdélníky.
- **Vztah** (relationship) zachycuje, jakým způsobem jsou dvě nebo více entit vztažené mezi sebou. Vztahy se označují slovesy, spojujícími dvě nebo více podstatných jmen. Příklad: vztah vlastní je mezi společností a počítačem,

vztah dohlíží je mezi zaměstnancem a oddělením, vztah hraje je mezi umělcem a písní, a vztah dokázal je mezi matematikem a matematickou teorií. Vztahy se zobrazují jako kosočtverce propojené čárami vedoucími ke každé entitě vztahu.

- Jak entity, tak i vztahy mohou obsahovat **atributy**. Např. entita zaměstnanec může obsahovat atribut rodné číslo (RČ); vztah dokázal může obsahovat atribut datum. Atributy se zobrazují jako elipsy propojené čarou s jejich vlastnící entitou. Každá entita (pokud nejde o slabou entitu) musí mít nejméně tolik atributů, aby byla za všech okolností jednoznačně identifikovatelná, jde o tzv. primární klíč.

Entity-relationship diagramy nezobrazují samostatné entity nebo jednoduché vztahy, zobrazují množiny entit a množiny vztahů. Např. určitá píseň je entita. Sbírka všech písní z databáze je množina entit. Vztah je sněden mezi dítětem a jeho obědem je jednoduchý vztah. Množina všech vztahů týkající se dětí a obědů v databázi je množina vztahů. [1i]

Chenova notace ER modelování používá obdélníky pro znázornění entit a kosočtverce pro znázornění vztahů. Tato notace je odpovídající, protože Chenovy vztahy jsou původně objekty – mohou samy obsahovat atributy i další vztahy. Existují však i jiné notace ER modelování. Zde uvádím některé další koncepce:

- IDEF1X,
- Bachmanova notace,
- Martinova notace,
- Crow’s Foot.

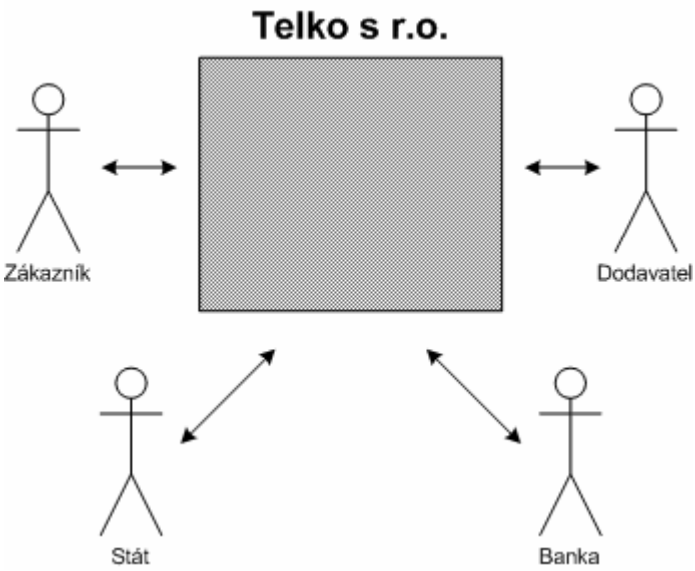
Pro tvorbu ER diagramů existuje na trhu celá řada softwarových nástrojů. Mezi placené nástroje spadají např. Avolution, ConceptDraw, ER/Studio, ERwin, DeZign for Databases, MEGA International, OmniGraffle, Oracle Designer, PowerDesigner, Rational Rose, RISE Editor, SmartDraw, Sparx Enterprise Architect, SQLyog, Toad Data Modeler, Microsoft Visio nebo Visual Paradigm. Já pro účely diplomové práce a tvorbu ER diagramů využiji bezplatného nástroje MySQL Workbench. [1i]

2.3 Současný zakázkový systém

V této části diplomové práce se zaměřím na analýzu současného zakázkového systému firmy Telko s r.o. Začnu s vymezením rozsahu systému, pokračovat budu přes úvodní studii systému a nakonec provedu porovnání silných a slabých stránek současného systému.

2.3.1 Vymezení rozsahu systému

Hned z úvodu se pokusím vymezit rozsah zakázkového systému firmy Telko s r.o., o kterém budu dále v rámci diplomové práce psát. Nejprve však vymezím hranice firmy jako takové. Kontext pro návrh firemních procesů lze zdokumentovat pomocí případu užití typu černá skříňka, kde jako zamýšlený systém vystupuje společnost nebo organizace. Obrázek 2-5 je toho názorným příkladem. Na tomto obrázku můžeme vidět černou skříňku (černě vyšrafovaný obdélník), která představuje firmu Telko s r.o. a čtyři subjekty (postavičky) reprezentující okolí firmy. To, co je uvnitř černé skříňky, je jádro podniku. Jádro podniku, je skryto před okolím a zahrnuje všechny firemní procesy. Z diagramu lze vyčíst, že černá skříňka je nějakým způsobem v kontaktu se zákazníkem, dodavatelem, státem a bankou. [1]



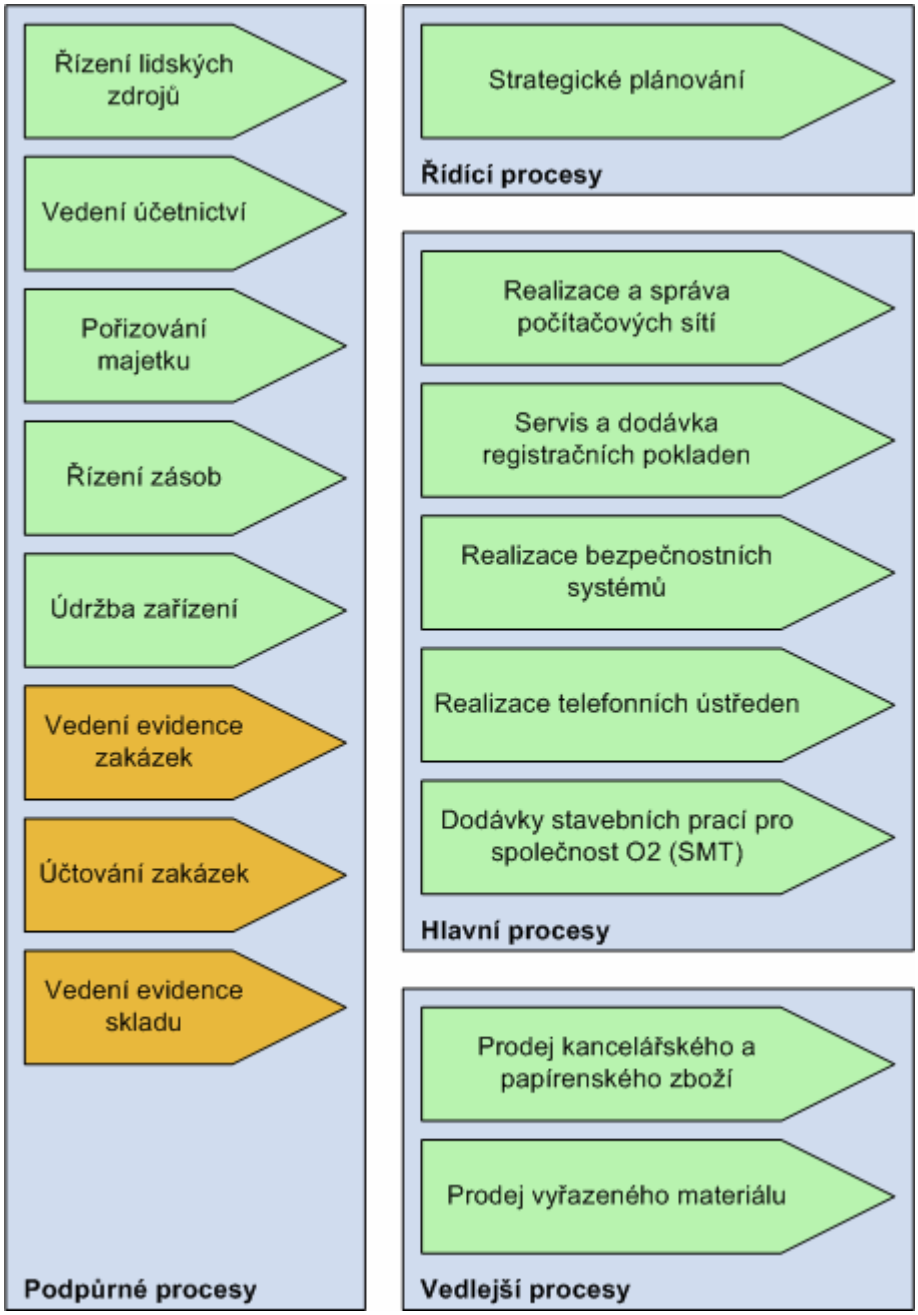
Obrázek 2-5: Diagram černé skříňky znázorňující kontext firmy Telko s r.o.

Ještě než přistoupím k popisu systému a jeho procesů, pokusím se nejprve stručně definovat pojem proces. **Proces lze obecně chápat jako logicky nebo chronologicky seřazený soubor činností s definovanými vstupy a výstupy, které vytvářejí ucelenou hodnotu pro zákazníka procesu**. Rozlišujeme dva typy zákazníků – externí a interní. **Zákazník externí** využívá výstupy organizace. Pro **zákazníka interního** platí totéž až na skutečnost, že on sám je subjektem uvnitř organizace. Dále podle kategorií zákazníků procesu můžeme procesy rozdělit do pěti kategorií. A to na:

- **procesy hlavní** – vytvářejí hodnotu pro externího zákazníka, jejich výsledkem je produkce takových výstupů, které zákazník požaduje; hlavní procesy jsou dány hlavními podnikatelskými činnostmi organizace, jež vedou k naplnění strategické vize a poslání podniku,
- **procesy vedlejší** – jsou obdobou hlavních procesů, avšak nejsou již pro firmu tolik důležité (nepodílejí se výrazným způsobem na hlavní činnosti podniku); jsou rovněž vykonávány pro externího zákazníka,
- **procesy řídící** – definují strategické cíle a způsoby zajištění jejich realizace v rámci celého podniku;
- **procesy podpůrné** – vytvářejí podmínky, které umožňují hlavní funkci procesů; vyznačují se tvorbou přidané hodnoty pro interního zákazníka;
- **procesy sdílené** – vytvářejí podmínky, které umožňují funkci všech procesů v rámci podniku; zpravidla mají pouze interního zákazníka.

Všechny z výše uvedených kategorií jsou součástí tzv. **rámcového modelu procesů**, kterým se budu dále zabývat. [7]

Rámcový model procesů firmy Telko s r.o., jehož kostra je znázorněna na následující stráně (viz obrázek 2-6), poukazuje na top-level procesy uvnitř organizace a jejich přiřazení k jednotlivým kategoriím, o kterých jsem se zmiňoval výše. Všechny procesy uvedené na obrázku 2-6 samozřejmě dekomponovat nebudu. V dalším textu diplomové práce bude má pozornost nejvíce směřovat k procesu ***vedení evidence zakázek***, procesu ***účtování zakázek*** a k procesu ***vedení evidence skladu***. Tyto tři procesy totiž tvoří základnu stávajícího i budoucího zakázkového systému.



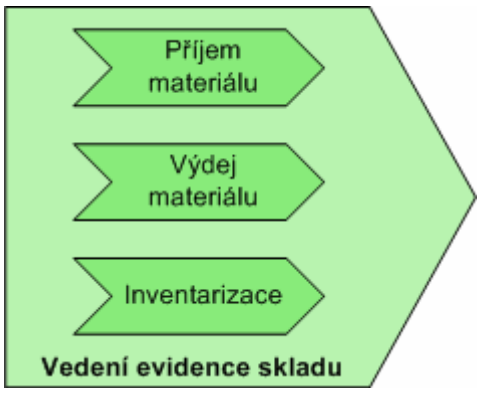
Obrázek 2-6: Rámcový model procesů firmy Telko s r.o.

Dekompozicí podpůrného procesu *vedení evidence zakázek* (viz obrázek 2-7, následující strana) zjistíme, jaké podprocesy tento top-level proces obsahuje. Proces *vedení evidence zakázek* zahrnuje tři podprocesy. Jsou jimi proces ***příjem zakázky***, proces ***modifikace zakázky*** a proces ***odstranění zakázky***.

Proces *příjem zakázky* je složen z řady činností. Jedná se např. o činnosti jako je kontrola zpráv poštovního klienta, kontaktování klienta pro zjištění podrobnějších informací, vložení zakázky do evidence apod. Proces *modifikace zakázky* zahrnuje činnosti jako je např. změna termínu zakázky, změna kontaktních údajů zákazníka, změna stavu zakázky (zda-li je hotová, vyúčtovaná, vyfakturovaná) apod. Proces *odstranění zakázky* pak obsahuje sekvenci činností vedoucí k odstranění zakázky ze systému.

Obrázek 2-7: Dekompozice procesu *vedení evidence zakázek*

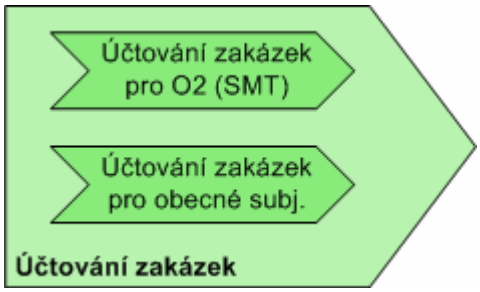
Dalším podpůrným procesem nejvyšší úrovně, o kterém se chci zmínit, je proces *vedení evidence skladu* materiálu. Tento proces lze rovněž dekomponovat na několik dílčích procesů. Jak zachycuje obrázek 2-8, jsou jimi proces ***příjem materiálu***, proces ***výdej materiálu*** a proces ***inventarizace***.



Obrázek 2-8: Dekompozice procesu *vedení evidence skladu*

Proces *příjem materiálu* lze charakterizovat činnostmi jako je vypsání předávacího protokolu, převzetí příjemky materiálu, fyzické uložení materiálu na skladu, zapsání patřičných údajů do evidence skladu apod. Proces *výdej materiálu* lze popsat obdobně jako proces příjmu materiálu avšak v opačném gardu. Proces *inventarizace* pak zahrnuje činnosti jako je zjišťování fyzického stavu materiálu na skladě a případné úpravy v evidenci materiálu s tím spojené.

Posledním z podpůrných top-level procesů, který se pokusím stručně charakterizovat, je proces *účtování zakázek*. Celý proces lze dekomponovat na dva dílčí procesy nižší úrovně. Na proces ***účtování zakázek pro O2 (SMT)*** a na proces ***účtování zakázek pro obecné subjekty***. Tato dekompozice je ilustrována na obrázku 2-9.



Obrázek 2-9: Dekompozice procesu účtování zakázek

Oba procesy jsou si do jisté míry podobné, avšak proces *účtování zakázek pro O2 (SMT)* zahrnuje několik činností navíc. Začnu tedy nejprve s odlehčenou variantou. Proces *účtování zakázek pro obecné subjekty* lze dále dekomponovat na tři procesy. Jedná se o proces ***přiřazení položek práce k zakázce***, ***přiřazení položek materiálu k zakázce*** a ***sestavení vyúčtování***. Proces *účtování zakázek pro O2 (SMT)* obsahuje navíc ještě dva další procesy. Jsou jimi proces ***sestavení schématu zakázky*** (grafické vyjádření jednotlivých prací) a proces ***vytvoření souhrnného vyúčtování***, jehož součástí je vyúčtování jednotlivých zakázek stejného typu. Vyúčtování se tedy neposílá pro každou zakázku zvlášť, ale posílá se nárazově – na vyžádání společnosti O2, resp. firmy SMT. Tento postup je zřejmě zaveden z důvodu úspory nákladů spojených s administrativou, jelikož firma Telko s r.o. realizuje velký počet relativně malých zakázek.

2.3.2 Úvodní studie současného systému zakázek

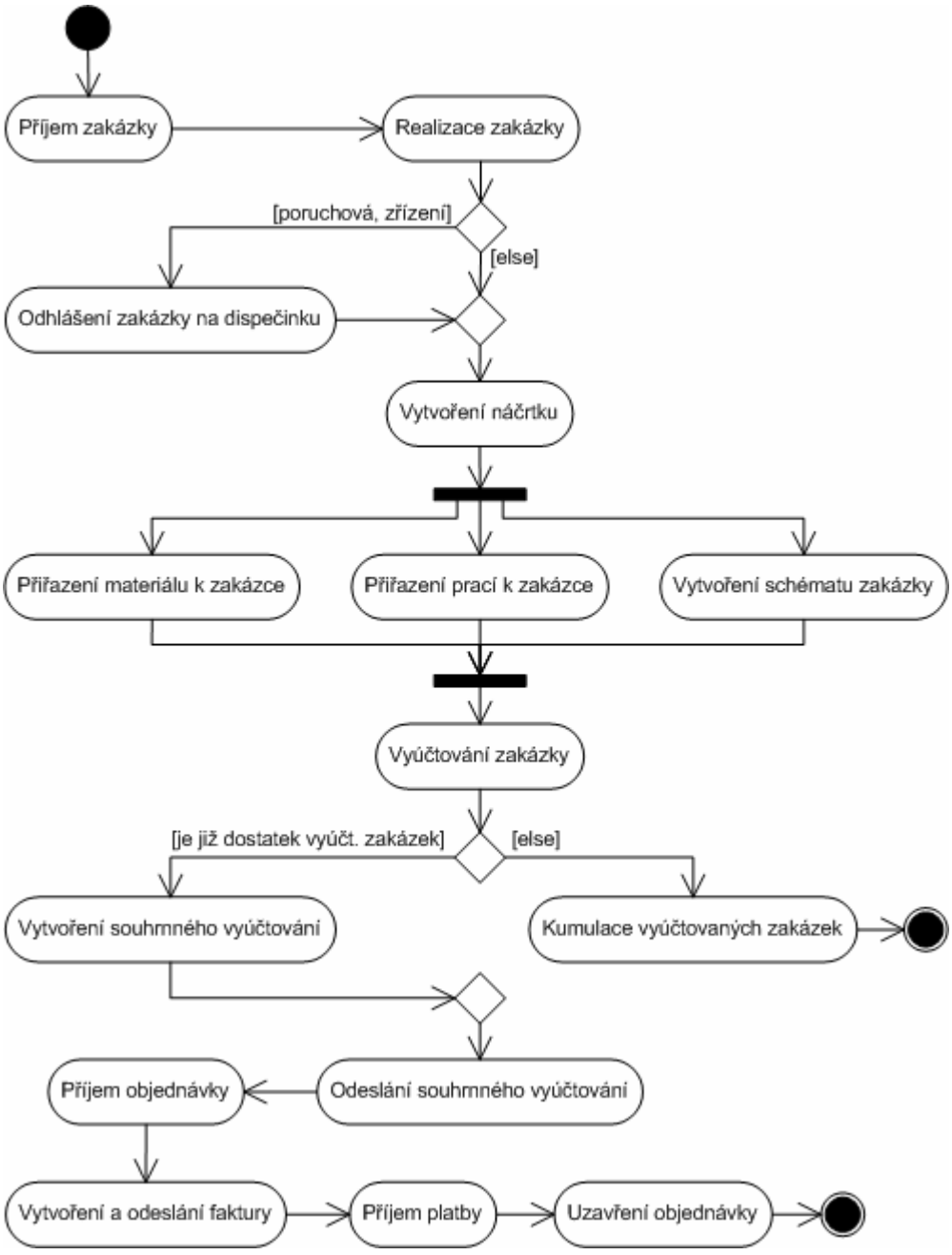
Pokud bych měl vybrat jedno jediné slovo, kterým bych charakterizoval současnou podobu zakázkového systému firmy Telko s r.o., bylo by tím slovem slovo útržkovitost. Slovo útržkovitost podle mého názoru nejlépe vystihuje současný systém zakázek ve firmě. Potřebné údaje, týkající se zakázek, jsou uloženy na různých místech, v různých formátech a v různém množství (výskyt duplicit). To platí jak pro fyzické dokumenty, tak i pro dokumenty v elektronické podobě. Snad jedinou výjimku tvoří e-mailový klient, kde jsou zprávy filtrovány a řazeny do složek podle jejich odesílatele a tak v něm lze zakázky snadno identifikovat. Tím, co jsem uvedl, nechci popírat existenci současného systému zakázek. Systém samozřejmě existuje. Neexistuje však k němu žádná bližší dokumentace. Kdybych měl využít modelu CMM (Capability Maturity Model), který byl vypracován organizací Software Engineering Institute, přiřadil bych firmu Telko s r.o. k prvnímu stupni vyspělosti, tedy k úvodní úrovni.

Úvodní úroveň modelu CMM lze charakterizovat jako výchozí a nachází se v ní všechny organizace, které **nemají své procesy definovány a formalizovány**. I přesto mohou na tomto stupni vyspělosti existovat organizace, které mají úspěšný management. Typické pro tento typ organizací je, že jsou problémy řešeny přístupem „ad-hoc“. Kvalita poskytovaných služeb a efektivita činností organizace je dosahována za cenu heroických výkonů jednotlivců v rámci dané organizace. Organizace na této úrovni vyspělosti kladou především důraz na získání a udržení kvalitních pracovníků, kteří definované a formalizované procesy považují za „osobní know-how“. [7]

Nyní se pokusím slovně popsat sekvenci činností, která následuje v současném systému po přijetí zakázky (objednávky). Tak tedy zákazník zavolá nebo zašle e-mail a sdělí tak firmě iniciály celé zakázky. Zakázka je zaevidována do systému (zde nemám na mysli softwarový systém nebo lépe řečeno databázovou aplikaci, která by zajistila integritu a konzistenci získaných dat – žádná podobná aplikace, která by evidenci zakázek zajišťovala, ve firmě zatím neexistuje). Následuje realizace zakázky firmou v dohodnutém termínu. Poté (výjimečně i během fáze realizace zakázky) zaměstnanci do stavebního deníku uvedou popis práce, kterou vykonali, uvedou materiál, který zastavěli a nakreslí náčrtek. Náčrtek se stavebním deníkem je předán administrativnímu zaměstnanci, jehož úkolem je tyto údaje zformalizovat. V nástroji MS Excel vytvoří seznam položek prací, které byly realizovány a seznam položek materiálu, který byl zastavěn. Poté pomocí

nástroje MS Visio překreslí náčrtek a vytvoří přehledné vyúčtování, které je spolu s fakturou zasláno zákazníkovi. Poté, co zákazník zaplatí, je zakázka uzavřena.

Na obrázku 2-10 je pomocí diagramu aktivit zachycena sekvence činností popsaná v předcházejícím odstavci. Musím však ještě podotknout, že je tento diagram přizpůsobený pro společnost SMT (viz proces účtování zakázek).



Obrázek 2-10: Diagram aktivit zakázkového systému

2.3.3 Identifikace silných a slabých stránek současného systému

Současný systém zakázek vzniknul na základě empirického přístupu a existuje v hlavách několika málo zaměstnanců firmy Telko s r.o. Lze jen stěží hovořit o silných stránkách systému, jehož dokumentace nikdy nebyla sestavena, avšak to, že současný systém není zdokumentován, ještě neznamená, že funguje nesprávným způsobem. Vlastní specifikaci současného systému zakázek jsem sestavil na základě pozorování jednotlivých činností v organizaci a po jejím prostudování bych chtěl v následující části upozornit na několik slabých míst.

- **Neexistence centrálního úložiště dat** – data jsou uložena nesystematicky na různých místech. Důsledkem toho jsou velké časové ztráty zapříčiněné např. hledáním potřebných dokumentů nebo podkladů pro účtování zakázek nebo každodenním zjišťováním aktuálnosti zakázek. Někdy se dokonce stává, že se na zakázku zapomene a pak se musí řešit po termínu (pokud tedy o to zákazník stále ještě stojí).
- **Výskyt duplicitních údajů –** stejná data se vyskytují na více místech. Ve svém důsledku tato skutečnost vnáší do systému zmatky a opticky se pak zdá být objem dat v systému větší než ve skutečnosti je.
- **Výskyt nekonzistentních údajů –** stejná data se zdají být jiná, než ve skutečnosti jsou (různé názvy stejných položek, hodnoty uložené v různých formátech, časový nesoulad uložených dat). Důsledkem toho je celkový zmatek v systému.
- **Neprovázanost procesu účtování zakázek a procesu vedení evidence skladu –** tím mám na mysli zejména fakt, že se při účtování zakázek přiřadí zakázkám seznam položek materiálu, ale na stavu skladu se to nijak neprojeví (alespoň tedy ne v rámci současné evidence). Výdejka materiálu se pak dělá dodatečně. Důsledkem jsou časové ztráty.
- **Neexistence specifikace současného systému zakázek** – je velmi složité automatizovat něco, o čem „nic“ nevíte. Pokud by existovala kvalitní dokumentace, vše by bylo jasnější. V důsledku tohoto problému může nastat situace, že po odchodu klíčových zaměstnanců nebude nikdo vědět, co má dělat.

3. Návrh databázové aplikace

V druhé kapitole diplomové práce budu svou pozornost věnovat především oblasti návrhu desktopové databázové aplikace. V této části Vám přiblížím svou vizi budoucího zakázkového systému firmy Telko s r.o. Poté se s vedoucím pracovníkem firmy pokusím sestavit specifikaci požadavků na budoucí systém. Dále se postupně zaměřím na návrh datové struktury aplikace, návrh a popis klíčových komponent, návrh grafického uživatelského rozhraní aplikace a na možnosti portability aplikace.

3.1 Vize budoucího zakázkového systému

Když se ještě vrátím k výčtu slabých stránek současného zakázkového systému, který jsem uvedl v předchozí kapitole, nabývám na přesvědčení, že jedním ze způsobů racionalizace stávajícího systému zakázek ve firmě Telko s r.o. je vytvoření nového systému, jehož jádrem bude databáze (databázová aplikace). Tento způsob racionalizace stávajícího systému není samozřejmě jedinou variantou, pomocí níž bychom mohli dosáhnout vyšší efektivity stávajícího systému zakázek ve firmě. Podle mého názoru je však tato cesta klíčovým aspektem optimalizace celého systému.

Má vize je asi taková. Chci vytvořit databázovou aplikaci, která bude zahrnovat několik modulů pro podporu tří firemních procesů popsaných v první kapitole. Těmito procesy jsou proces *vedení evidence zakázek*, proces *vedení evidence skladu* a proces *účtování zakázek*. Aplikace by tedy měla obsahovat modul pro evidenci skladu, modul pro evidenci a účtování zakázek a modul pro správu položek materiálu a prací. Zároveň bych chtěl sestavit aplikaci, která by byla přehledná, intuitivní (jednoduchá na ovládání), bezpečná a zároveň plně funkční. Nejvíce ze všeho bych si ale přál, aby vytvořená aplikace byla v očích zákazníka vnímána pozitivně a měla pro něj smysl.

Původně jsem chtěl vytvořit grafický nástroj („něco“ na styl MS Visio), jehož pomocí by uživatel kreslil pouze schémata zakázek („přetahoval myší objekty“) a seznam položek prací spolu se seznamem zastavěného materiálu by se generoval automaticky. Tímto způsobem by uživatel spojil tři činnosti v jednu a ušetřil by tak značné množství času. Nicméně výše popsaná varianta se mi po bližším prostudování zdála poněkud komplikovaná a tak zřejmě nakonec zůstane pouze v mých představách.

3.2 Specifikace uživatelských požadavků

Svou vizi budoucího systému zakázek jsem představil vedoucímu pracovníkovi firmy a na jejím základě byla sestavena specifikace uživatelských požadavků. Stručný přehled uživatelských požadavků uvádím v tabulce 3.1. Není vyloučeno, že během vývoje aplikace nedojde k modifikaci uživatelských požadavků. Z tohoto důvodu, budu aplikaci vyvíjet inkrementálním přístupem a zákazníkovi bude dodávána po částech ve zhruba dvoutýdenních cyklech. To znamená, že každá část bude rozšířením nebo úpravou části předchozí a bude testována se zákazníkem tak, aby maximálně vyhovovala jeho potřebám. Využitím tohoto přístupu zákazník uvidí, že se na jeho systému neustále pracuje a rovněž se touto cestou výrazně sníží riziko odmítnutí zařazení softwaru do provozu.

	Požadavek	Priorita
1.	Vložení zakázky do systému	1
2.	Modifikace zakázky	1
3.	Odstranění zakázky	1
4.	Vytvoření přehledného zobrazení zakázek	3
5.	Možnost filtrování zakázek	2
6.	Možnost vložení přílohy k zakázce	4
7.	Tisk stručného přehledu zakázky	3
8.	Vytvoření dialogu pro účtování zakázek (přiřazení položek materiálu a položek práce k zakázce)	1
9.	Možnost vytvořit souhrnné vyúčtování odpovídající normám společnosti SMT	1
10.	Možnost vytvořit vyúčtování pro obecné subjekty	2
11.	Možnost exportu souhrnného vyúčtování do EXCELU	1
12.	Vytvoření přehledného zobrazení skladu	1
13.	Možnost vložit příjemku materiálu	1
14.	Možnost vložit výdejku materiálu	1
15.	Možnost zobrazení stavu skladu materiálu	1
16.	Možnost exportu stavu skladu materiálu do EXCELU	1
17.	Možnost exportu obsahu příjemky nebo výdejky materiálu do EXCELU	2
16.	Možnost uživatelské editace položek materiálu	3
17.	Možnost uživatelské editace položek práce	3
18.	Možnost přiřazení položek materiálu k uživatelsky vytvořeným skupinám	2
19.	Možnost přiřazení položek práce k uživatelsky vytvořeným skupinám	2
20.	Atraktivní uživatelské rozhraní	4

Tabulka 3.1: Specifikace uživatelských požadavků

3.3 Návrh datové struktury aplikace

Na základě analýzy specifikace uživatelských požadavků, se pokusím identifikovat entity a jejich atributy, bez kterých by funkcionalita zakázkového systému nebyla možná. Vybrané entity pak budou představovat základ datové struktury aplikace.

Strukturu nejdůležitějších entit celého systému jsem vyjádřil pod tímto odstavcem v podobě několika tabulek. Teď se pokusím formát těchto tabulek blíže specifikovat. Význam polí *Atribut* a *Datový typ* je zřejmý. Pole *Null* nabývá dvou hodnot. Pokud je hodnota pole *Null* rovna *Ne*, znamená to, že daný atribut nesmí být *Null*, tedy musí být vždy vyplněn. Ve druhém případě platí, že atribut nemusí být vyplněn. Pole *Klíč* může obsahovat tři hodnoty. Tyto hodnoty mohou být *PK* (primární klíč), *FK* (cizí klíč) a nebo může být toto pole prázdné (nemá klíč). V poli *Poznámka* je pak uveden případný popis, který se k danému atributu váže. Nyní již samotný přehled klíčových entit.

Entita *zakazka*				
Atribut	**Datový typ**	**Klíč**	**Null**	**Poznámka**
id	INT(10)	PK	Ne	
id_o2	VARCHAR(30)		Ne	Identifikátor zakázky dle specifikace firmy O2
id_typ_zakazky_fk	INT(10)	FK	Ne	Odkaz do tabulky typů zakázek
id_stav_zakazky_fk	INT(10)	FK	Ne	Odkaz do tabulky stavů zakázek
datum_zaevidovani	DATETIME		Ano	Datum vložení zakázky do systému
datum_posledni_editace	TIMESTAMP		Ne	
termin_dokonceni	DATETIME		Ne	Datum do kdy má být zakázka realizována
ranzirovani	TINYINT(1)		Ne	Specifická činnost prováděná na ústředně
lokalita	VARCHAR(50)		Ne	
slovni_popis	VARCHAR(5000)		Ne	
id_firma	INT(10)	FK	Ne	Odkaz do tabulky firem
id_tarif	INT(10)	FK	Ne	Odkaz do tabulky cenových tarifů
vfm	VARCHAR(30)		Ano	Id. přidělený po odhlášení zakázky na dispečinku

Tabulka 3.2: Struktura entity *zakazka*

Entita *prace_polozka*				
Atribut	**Datový typ**	**Klíč**	**Null**	**Poznámka**
id_sap	INT(10)	PK	Ne	Id. dle specifikace firmy O2
popis	VARCHAR(100)		Ne	
merna_jednotka	VARCHAR(3)		Ne	
cena_stand	FLOAT		Ne	Cena standard
cena_exp	FLOAT		Ne	Cena expres
cena_por	FLOAT		Ne	Cena poruchy
cena_firma	FLOAT		Ano	Cena firemní
kategorie	VARCHAR(10)	FK	Ne	Odkaz do tabulky kategorie práce
smt_nazev	VARCHAR(10)		Ne	Specifický název položky materiálu dle firmy SMT

Tabulka 3.3: Struktura entity *prace_polozka*

Entita *material_polozka*				
Atribut	**Datový typ**	**Klíč**	**Null**	**Poznámka**
id_sap	INT(10)	PK	Ne	Id. dle specifikace firmy O2
id_sap_stare	INT(10)		Ano	Id. dle staré specifikace firmy O2
popis	VARCHAR(100)		Ne	
merna_jednotka	VARCHAR(3)		Ne	
id_kategorie_fk	VARCHAR(10)	FK	Ne	Odkaz do tabulky kategorie materiálu

Tabulka 3.4: Struktura entity *material_polozka*

Entita *vyuctovani*				
Atribut	Datový typ	Klíč	Null	Poznámka
id	INT(10)	PK	Ne	
nazev	VARCHAR(45)		Ne	Název vyúčtování
datum	DATETIME		Ne	Datum vytvoření vyúčt.
poznamka	VARCHAR(100)		Ano	Poznamka k vyúčtování
id_dodavatel_fk	INT(10)	FK	Ne	Odkaz do tabulky firem, implicitně nastaveno na firmu Telko s r.o.
id_odberatel_fk	INT(10)	FK	Ne	Odkaz do tabulky firem
datum_posledni_aktualizace	TIMESTAMP		Ne	
id_vydejka_mat_fk	INT(10)	FK	Ne	Odkaz do tabulky příjemek a výdejek materiálu
cislo_objednavky	VARCHAR(45)		Ano	Číslování objednávek pro obecné subjekty

Tabulka 3.5: Struktura entity *vyuctovani*

Entita *firma*				
Atribut	**Datový typ**	**Klíč**	**Null**	**Poznámka**
id	INT(10)	PK	Ne	
nazev	VARCHAR(45)		Ne	Název firmy
adresa	VARCHAR(100)		Ano	Adresa firmy
ic	VARCHAR(45)		Ano	IČ firmy
dic	VARCHAR(45)		Ano	DIČ firmy
tel_kontakt	VARCHAR(45)		Ano	Telefonní kontakt na firmu

Tabulka 3.6: Struktura entity *firma*

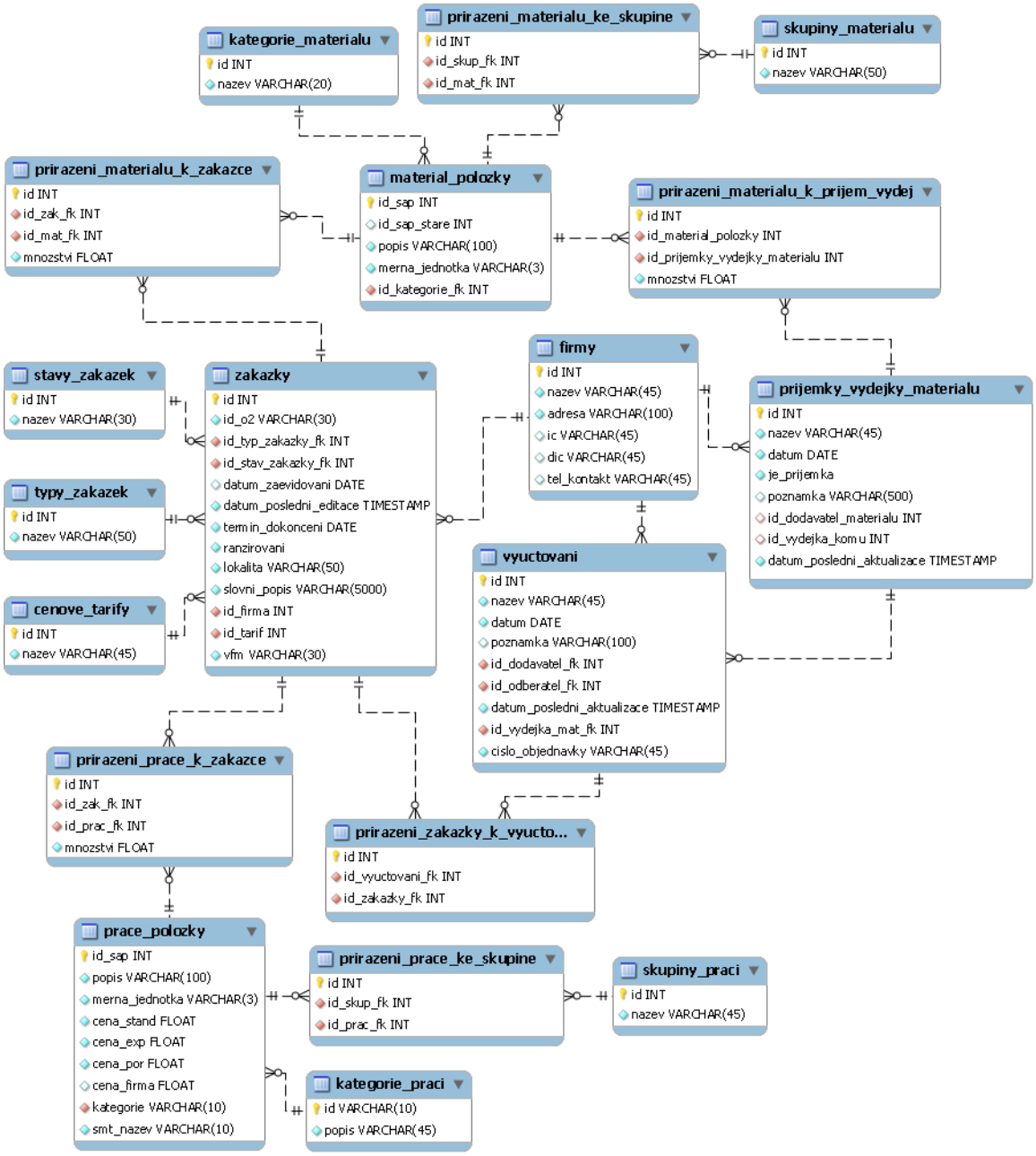
Entita *prijemka_vydejka_materialu*				
Atribut	**Datový typ**	**Klíč**	**Null**	**Poznámka**
id	INT(10)	PK	Ne	
nazev	VARCHAR(45)		Ne	Název příjemky nebo výdejky materiálu
datum	DATETIME		Ne	Datum vytvoření příj. nebo výd. mat.
je_prijemka	TINYINT(1)		Ne	0 = výdejka; 1 = příjemka
poznamka	VARCHAR(500)		Ano	
id_dodavatel_materialu	INT(10)	FK	Ano	Odkaz do tabulky firem
id_vydejka_komu	INT(10)	FK	Ano	Odkaz do tabulky firem
datum_posledni_aktualizace	TIMESTAMP		Ne	

Tabulka 3.7: Struktura entity *prijemka_vydejka_materialu*

Entita *prirazeni_materialu_k_prijem_vydej*				
Atribut	**Datový typ**	**Klíč**	**Null**	**Poznámka**
id	INT(10)	PK	Ne	
id_material_polozky	INT(10)		Ne	Odkaz do tabulky materiálu
id_prijemky_vydejky_materialu	INT(10)		Ne	Odkaz do tabulky příjemek a výdejek materiálu
mnozstvi	FLOAT		Ne	Přiřazené množství materiálu k příjemce nebo výdejce materiálu

Tabulka 3.8: Struktura entity *prirazeni_materialu_k_prijem_vydej*

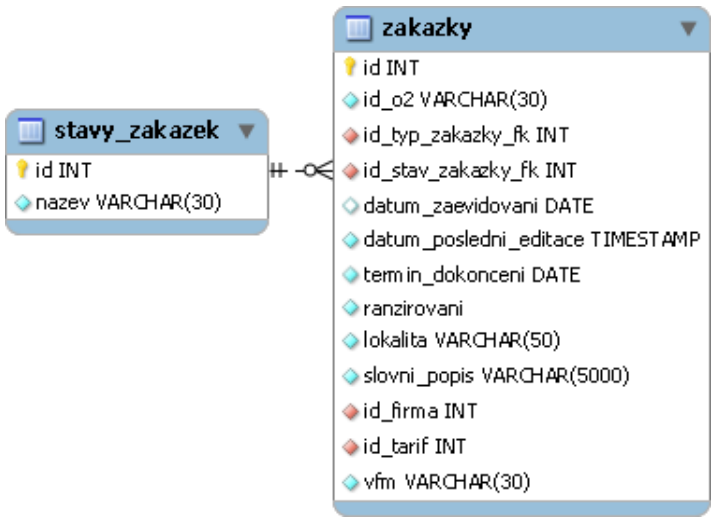
Pokud bych měl dále pokračovat a vyjádřit tak kompletní výčet entit, popsal bych tím ještě několik stran papíru a jistě mi dáte za pravdu, že by to bylo celkem zbytečné. Proto se teď zaměřím na propojení entit v komplexní systém. Toto propojení jsem zachytil v ER diagramu (viz obrázek 3-1), který jsem vytvořil pomocí nástroje MySQL Workbench.



Obrázek 3-1: ER diagram (notace Crow's Foot)

Vrátím-li se k charakteristice jednotlivých entit, chtěl bych ještě uvést několik poznámek k jejich hodnotám a relacím. Defaultní hodnotou všech atributů, jež mohou zůstat nevyplněné, je hodnota *null*. U atributů, u kterých se očekávají hodnoty 0 nebo 1 (datový typ *tinyint* nebo *boolean*), bude jako výchozí hodnota nastavena 0 nebo hodnota *false*. Hodnota primárních klíčů bude generována automaticky, přičemž počáteční hodnotou je 1 a každý další záznam bude mít hodnotu primárního klíče o jedničku vyšší (viz funkce *auto increment*). Primární a sekundární klíče slouží k provázání a organizaci dat v databázi, pro uživatele jsou zcela transparentní.

Na obrázku 3-1 můžeme vidět propojení jednotlivých entit. Spojnice dvou entit je nazývána relací. Relace lze rozčlenit do několika kategorií podle jejich typu. Klíčovým faktorem je z tohoto hlediska počáteční a koncový symbol dané relace. Já jsem pro vytváření ER diagramu a de facto i budování databáze, jak již bylo uvedeno, použil nástroj MySQL Workbench. Tento nástroj za mě vygeneroval i jednotlivé relace. V mém ER diagramu lze mezi entitami rozpoznat pouze dva typy ukončovacích symbolů. Buď se jedná o symbol dvou čárek, nebo o symbol kolečka a jakési vidličky z ní vycházející. (viz názorný příklad ilustrovaný obrázkem 3-2).



Obrázek 3-2: Příklad relace mezi entitami

Obrázek 3-2 lze interpretovat tak, že každá zakázka má právě jeden stav. Z opačného pohledu pak jeden (stejný) stav zakázky může mít (vlastnit) žádná nebo více zakázek.

3.4 Návrh a popis klíčových komponent aplikace

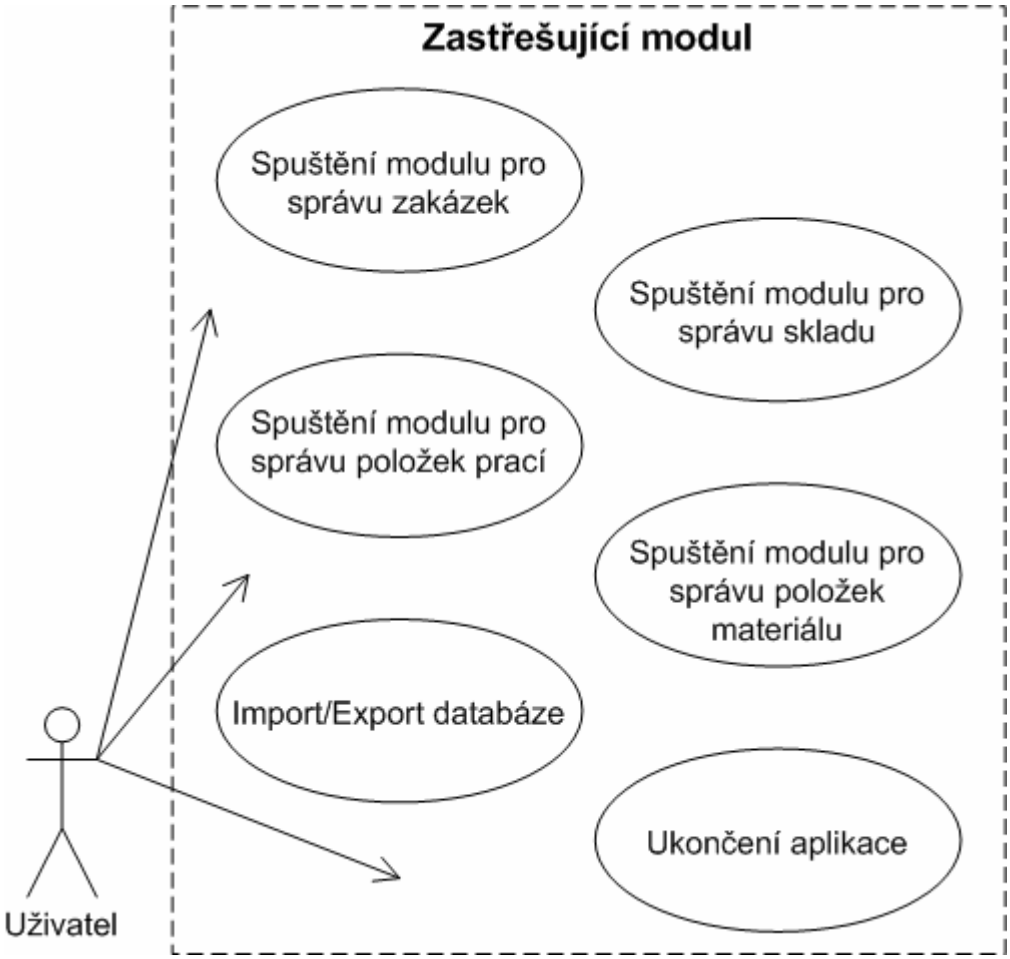
Při návrhu a popisu jednotlivých komponent databázové aplikace budu vycházet ze specifikace uživatelských požadavků, kterou jsem uvedl na začátku kapitoly. Celou aplikaci se pokusím logicky rozčlenit tak, aby každá část (modul) pokrývala určité spektrum funkcionalit, jež jsou požadovány zákazníkem.

Architektura aplikace bude klient-server (dvouvrstvá), přičemž do části klienta zahrnu jak business logiku, tak i vrstvu prezentační – hovořím tedy o tlustém klientovi. Z důvodu „jednoduchosti“ aplikace a faktoru, že je aplikace určena pro jednoho uživatele, nevyužiji zatím služeb žádného aplikačního serveru (např. AS Glassfish, AS JBoss apod.). Pokud jsem napsal, že se bude jednat o architekturu klient-server, pak na straně serveru bude stát databázový server, který bude provozován na stejném počítači spolu s klientem. Ptáte se proč? Prvním důvodem je fakt, že se jedná o jednouživatelskou aplikaci, jak již bylo uvedeno. Druhým, neméně důležitým, důvodem je bezpečnost. Tím však nechci říct, že aplikace nebude portabilní. O možnostech portability aplikace budu hovořit dále v textu. Pokud by někdy v budoucnu nastal požadavek na zpřístupnění databázového serveru přes internet, nemělo by to znamenat žádný větší problém. Nyní již přejdu k návrhu jednotlivých modulů.

3.4.1 Zastřešující modul

Zastřešující modul, jako jediný, neřeší žádný z uživatelských požadavků. Jeho smyslem je zastřešovat jednotlivé moduly a jeho funkcionalita (z pohledu uživatele) je popsána na následující straně pomocí diagramu případů užití (viz obrázek 3-3). Z ilustrace lze vyčíst všechny funkcionality (případy užití), které má uživatel v tomto modulu k dispozici. Konkrétně jsou to případy užití *spuštění modulu pro správu zakázek*, spuštění *modulu pro správu skladu*, *spuštění modulu pro správu položek prací*, *spuštění modulu pro správu položek materiálu*, *import/export databáze* a *ukončení aplikace*.

Význam všech výše uvedených případů užití je intuitivní. Za zmínku stojí snad jen případ užití *import/export databáze*. Pokud uživatel využije této funkcionality, zobrazí se mu nabídka, kde buď zvolí možnost pro exportování databáze do souboru (na výběr bude mít různé formáty – *.sql*, *.csv* a *.xml*), nebo opačnou možnost pro import databáze ze souboru.



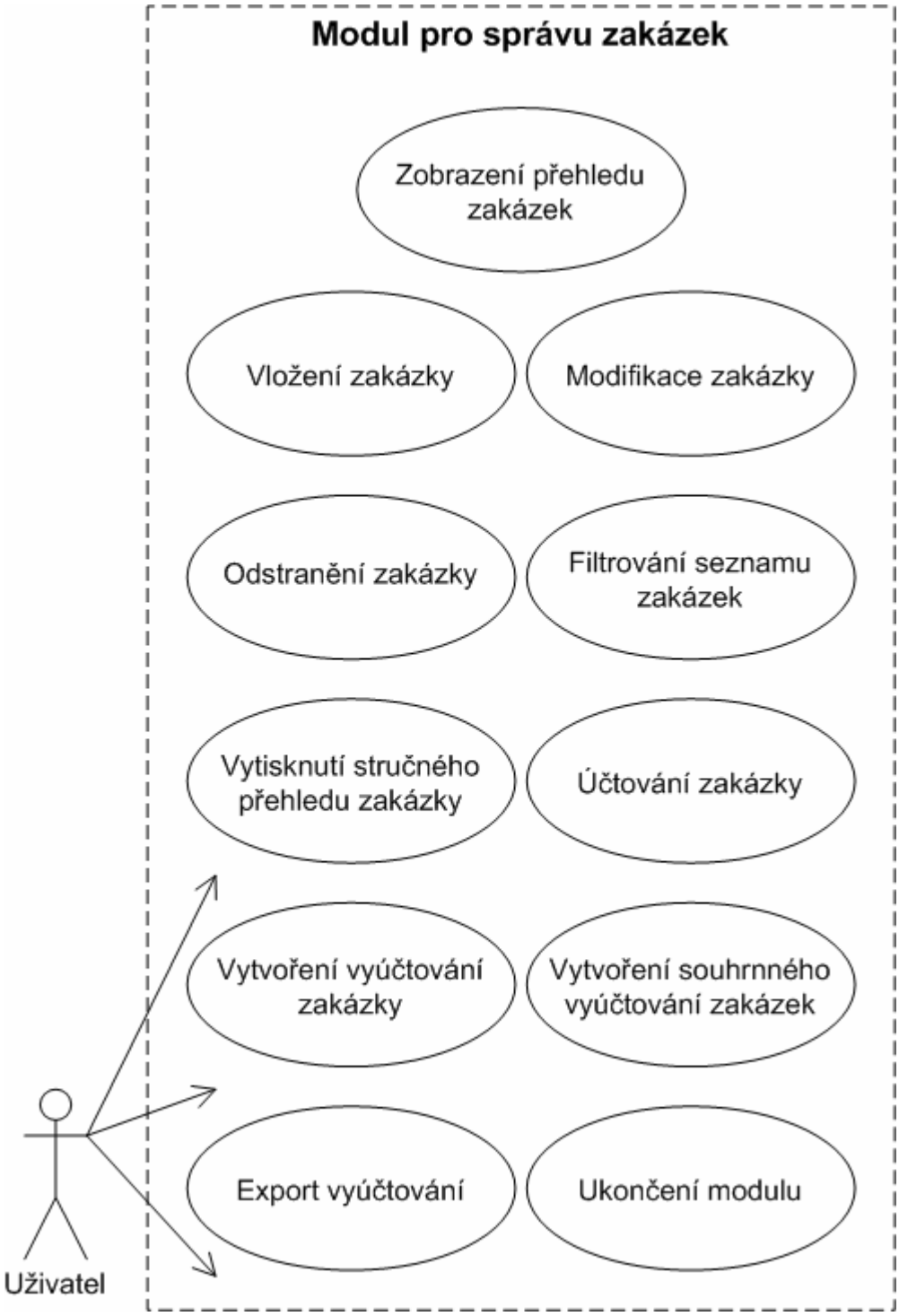
Obrázek 3-3: Diagram případů užití znázorňující funkcionalitu *zastřešujícího modulu*

Důležitou úlohou *zastřešujícího modulu* aplikace, která však bude z hlediska uživatele zcela transparentní, je vytvoření konektivity na databázi a její následné uložení do *veřejné statické* proměnné (kvůli maximální dostupnosti z libovolného místa budoucí databázové aplikace). Technologii tzv. *connection poolingu* ve své aplikaci zatím nevyužiji a to zejména z důvodů, které jsem již uvedl dříve (viz jednouživatelská aplikace).

Dále by do tohoto modulu měla být zahrnuta funkce pro customizaci softwaru potřebám zákazníka. Tím mám konkrétně na mysli např. nastavení výchozích URL cest, nastavení velikosti písma, nastavení barevných motivů, definování uživatelských filtrů, aby při dalším spuštění aplikace byly nastaveny dle potřeb uživatele, ruční modifikace sazeb (DPH, jiné cenové přirážky) apod.

Z uživatelského hlediska realizuje *zastřešující modul* jakýsi rozcestník. Je na uživateli, kterou cestou se v programu dále vydá.

3.4.2 Modul pro správu zakázek



Obrázek 3-4: Diagram případů užití znázorňující funkcionalitu *modulu pro správu zakázek*

Na obrázku 3-4, zobrazeným na předchozí straně, můžeme vidět výčet funkcí, které tento modul pokrývá. Nyní se pokusím jednotlivé funkce (případy užití) popsat.

Případ užití *zobrazení přehledu zakázek* je tím prvním, co uživatel vyvolá při přechodu ze *zastřešujícího modulu* do *modulu pro správu zakázek*. Případ užití, nebo chcete-li use case, *zobrazení přehledu zakázek* ve své podstatě načte soubory dat z tabulek databázového serveru a v patřičné formě je pak uloží do klientské aplikace (do jeho proměnných hodnot). Data, získaná tímto způsobem, jsou pak výchozím zdrojem při zobrazování jednotlivých tabulek. Funkce *zobrazení přehledu zakázek* je uživatelem vyvolána i v případě, kdy provádí změny v systému (např. vytvoření nové zakázky, odstranění stávající zakázky apod.).

Případ užití *vložení zakázky* do systému je základním požadavkem na funkcionalitu, bez kterého by celý systém nemohl účinně fungovat. Textová specifikace tohoto případu užití je uvedena v tabulce 3.9. Tabulka je rozdělena do čtyř částí. V horní části je uveden název případu užití. Poté následuje úroveň cíle, hlavní úspěšný scénář a nakonec jeho rozšíření (rizika a jejich opatření).

UC - Vložení zakázky
Úroveň cíle: úroveň mořské hladiny (uživatelský cíl)
Hlavní úspěšný scénář: 1. Uživatel zvolí příkaz pro vytvoření nové zakázky 2. Systém zobrazí vstupní dialog pro vytvoření nové zakázky 3. Uživatel vyplní potřebné informace a nakonec zvolí příkaz pro potvrzení 4. Systém zkontroluje konzistenci a správnost vstupních dat 5. Systém uloží zakázku do systému a ukončí dialog 6. Systém promítne změny do klientské aplikace
Rozšíření: 2a: Nenačtou se potřebné údaje z databáze (např. seznam firem) 1. Systém upozorní uživatele o chybě databázového serveru a ukončí dialog 4a: Nejsou vyplněna povinná pole 1. Systém upozorní uživatele o nevyplněných polích, pak návrat do kroku 3 5a: Chyba na straně databázového serveru 1. Systém upozorní uživatele, vypíše údaje o chybě a ukončí dialog

Tabulka 3.9: Textová specifikace případu užití *vložení zakázky*

Modifikace zakázky je obdobou případu užití *vložení zakázky* do systému. Jediným rozdílem je fakt, že se nevytváří zakázka nová, nýbrž se upraví údaje zakázky stávající. Když uživatel provede příkaz pro modifikaci zvolené zakázky, otevře systém dialogové okno, které je již implicitně vyplněno daty z již existující zakázky. Poté následuje scénář totožný se scénářem vložení zakázky (počínaje krokem 3, viz tabulka 3.9).

Případ užití *odstranění zakázky* ze systému nejen že smaže samotnou zakázku, ale spolu s ní zároveň odstraní veškeré vazby a data s ní související (i z jiných tabulek). Z toho důvodu systém při pokusu o smazání zakázky vyzve uživatele, aby zvážil danou skutečnost a přesvědčil se, zda ví, co dělá. Pokud uživatel chce, aby zakázka pouze zmizela ze systému a integrita dat zůstala zachována, bude moci nastavit stav zakázky na *neaktivní*.

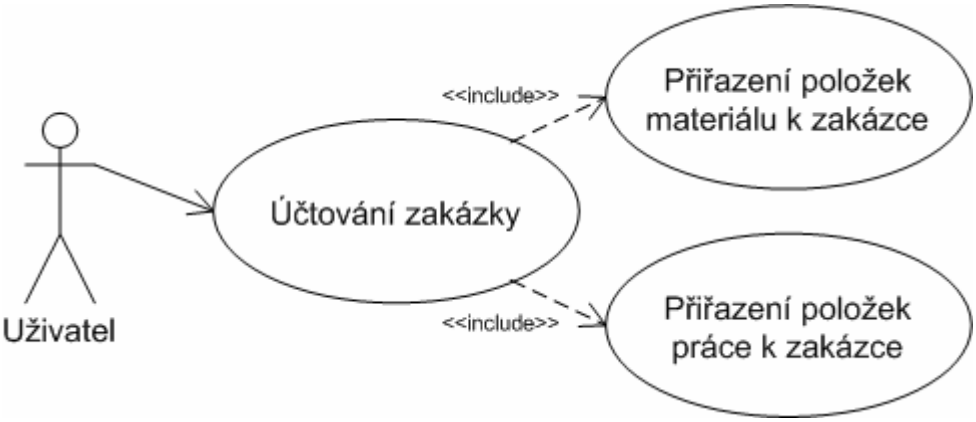
Filtrování zakázek je další nedílnou součástí specifikace uživatelských požadavků. V klientské aplikaci bude moci uživatel filtrovat podle několika kritérií. Tato kritéria vznikla na základě bližší specifikace zákazníkem. Komponenta pro filtrování by měla být navržena tak, aby byla snadno a intuitivně použitelná a zároveň aby nezabírala příliš mnoho prostoru v okně desktopové aplikace. Kritéria pro filtrování by měla zahrnovat:

- možnost zobrazení všech zakázek,
- možnost hledání zakázek podle atributu *id_o2*,
- možnost využít uživatelský filtr, ve kterém by bylo možno kombinovat kritéria jako je typ zakázky, stav zakázky, lokalita zakázky, datum vložení zakázky do systému (nastavení dolní hranice, horní hranice nebo intervalu).

Po zadání patřičných kritérií se budou záznamy zakázek řadit do seznamu podle termínu jejich dokončení tak, že zakázka, která má být vyhotovena nejdříve, bude uvedena jako první.

Pokud bude chtít uživatel tisknout přehled zvolené zakázky, využije případu užití *vytisknutí stručného přehledu zakázky*. Když uživatel provede příkaz určený pro tisk, systém zobrazí tiskový dialog s vlastnostmi tisku (např. výběr tiskárny, černobílý tisk, barevný tisk apod.). Po potvrzení tiskového dialogu proběhne tisk.

Případ užití *účtování zakázky* je již o něco více komplikovaný, proto se mu teď budu věnovat podrobněji. Nejprve provedu dekompozici případu užití tak, jak uvádím na následující straně na obrázku 3-5.



Obrázek 3-5: Diagram případů užití znázorňující dekompozici *účtování zakázky*

Dle ilustrace zachycené na obrázku 3-5 je zřejmé, že případ užití *účtování zakázky* v sobě zahrnuje (<<*include*>>) další dva případy užití, a to *přiřazení položek materiálu k zakázce* a *přiřazení položek práce k zakázce*. Díky dekompozici dosáhnu vyšší čitelnosti případu užití *účtování zakázky*. Textová specifikace tohoto případu užití bude nyní vypadat následovně (viz tabulka 3.10).

UC – Účtování zakázky
Úroveň cíle: úroveň mořské hladiny (uživatelský cíl)
Hlavní úspěšný scénář: 1. Uživatel zvolí příkaz pro účtování zakázky 2. Systém zobrazí potřebný dialog 3. Uživatel přiřadí položky materiálu k zakázce 4. Uživatel přiřadí položky práce k zakázce 5. Uživatel označí zakázku jako vyúčtovanou 6. Systém uloží zadaná data do systému a ukončí dialog 7. Systém promítne změny do klientské aplikace
Rozšíření: 2a: Nenačtou se potřebné údaje z databáze (např. seznam materiálu) 1. Systém upozorní uživatele o chybě databázového serveru a ukončí dialog 6a: Chyba na straně databázového serveru 1. Systém upozorní uživatele, vypíše údaje o chybě a ukončí dialog

Tabulka 3.10: Textová specifikace případu užití *účtování zakázky*

Tabulka 3.10 je na první pohled formálně totožná s tabulkou 3.9, která je uvedená na straně 42. Mezi těmito tabulkami však existuje jeden rozdíl. Rozdíl spočívá v tom, že případ užití *účtování zakázky* v sobě zahrnuje, jak již bylo zmíněno, další dva konkretizující případy užití. V tabulce 3.10 je tato skutečnost reprezentována podtržením textu v hlavním úspěšném scénáři (viz krok 3 a 4). Samotné podtržení by mělo naznačovat hyperodkaz. Syntaxe nám tedy říká, že by k tomuto případu užití měly připadat ještě další dva konkretizující případy užití. Já provedu textovou specifikaci pouze jednoho z nich, protože jsou si oba velmi podobné. V tabulce 3.11 uvádím textovou specifikaci případu užití *přiřazení položek materiálu k zakázce*.

UC – Přiřazení položek materiálu k zakázce
Úroveň cíle: úroveň mořské hladiny (uživatelský cíl)
Hlavní úspěšný scénář: 1. Uživatel zvolí příkaz pro přiřazení položek materiálu k zakázce 2. Systém zobrazí potřebný dialog 3. Uživatel vybere potřebné položky, nastaví jejich množství a potvrdí jejich přiřazení je k zakázce 4. Systém uloží zadaná data do systému a ukončí dialog 5. Systém promítne změny do klientské aplikace
Rozšíření: 2a: Nenačtou se potřebné údaje z databáze (např. seznam materiálu) 1. Systém upozorní uživatele o chybě databázového serveru a ukončí dialog 3a: Uživatel nastaví u položky vybraného materiálu množství na hodnotu menší než 0 1. Systém upozorní uživatele o nevhodném vyplnění pole a přiřazení této položky materiálu k zakázce se stornuje 4a: Chyba na straně databázového serveru 1. Systém upozorní uživatele, vypíše údaje o chybě a ukončí dialog

Tabulka 3.11: Textová specifikace případu užití *přiřazení položek materiálu k zakázce*

Ze strany zákazníka vznikl nový požadavek na možnost filtrování v části programu, kde uživatel provádí přiřazování položek materiálu a práce. Důraz byl kladen zejména na filtrování položek podle atributů *id_sap* a *popis*.

Nyní již přejdu k velmi stručné charakteristice zbylých případů užití, které jsou součástí modulu určenému pro správu zakázek.

Vytvoření vyúčtování zakázky a *vytvoření souhrnného vyúčtování zakázky* se bude realizovat v samostatném dialogu. Oba procesy jsou téměř totožné, proto uvedu stručnou textovou specifikaci toho složitějšího. *Vytvoření souhrnného vyúčtování zakázky* probíhá tak, jak znázorňuji v tabulce 3.12.

UC – Vytvoření souhrnného vyúčtování zakázky
Úroveň cíle: úroveň mořské hladiny (uživatelský cíl) **Hlavní úspěšný scénář:** 1. Uživatel zvolí příkaz pro vytvoření souhrnného vyúčtování zakázky 2. Systém zobrazí potřebný dialog 3. Uživatel vybere zakázky, které jsou již vyúčtovány a přiřadí je k souhrnnému vyúčtování 4. Uživatel vyplní patřičné identifikační údaje (název, datum, odběratelský subjekt apod.) a potvrdí vytvoření souhrnného vyúčtování 5. Systém zkontroluje konzistenci a správnost vstupních dat 6. Systém uloží zadaná data do systému (mimo jiné vytvoří výdejku materiálu) a ukončí dialog 7. Systém promítne změny do klientské aplikace **Rozšíření:** 2a: Nenačtou se potřebné údaje z databáze (např. seznam zakázek) 1. Systém upozorní uživatele o chybě databázového serveru a ukončí dialog 5a: Nejsou vyplněna povinná pole 1. Systém upozorní uživatele o nevyplněných polích, pak návrat do kroku 4 6a: Chyba na straně databázového serveru 1. Systém upozorní uživatele, vypíše údaje o chybě a ukončí dialog (využití transakce)

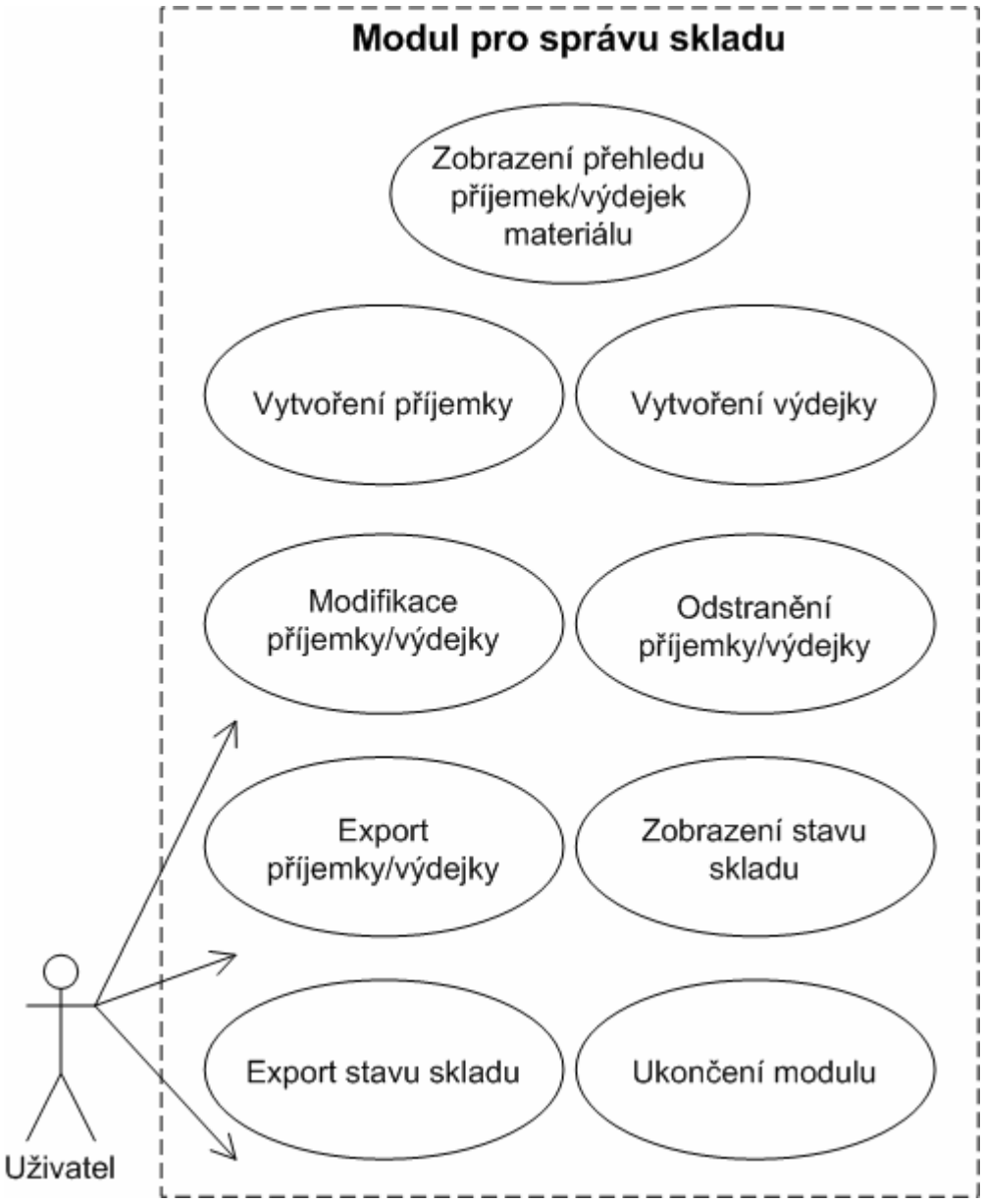
Tabulka 3.12: Textová specifikace případu užití *vytvoření souhrnného vyúčtování zakázky*

Vytvoření vyúčtování zakázky (klasické) by proběhlo obdobně. Rozdíl tvoří pouze fakt, že uživatel bude moci zvolit maximálně jednu vyúčtovanou zakázku.

Finální etapou celého modulu je případ užití *export vyúčtování*. Tento use case lze popsat poměrně jednoduše. Uživatel vybere vyúčtování ze seznamu (klasické nebo souhrnné) a zvolí příkaz pro export. Systém uživateli zobrazí nabídku možností exportu (typ souboru, cesta). Nejvyšší prioritu má z pohledu zákazníka export dat do formátu *.xls* (tedy do aplikace MS Excel). Formáty *.xml* a *.csv* mají zatím prioritu nižší.

3.4.3 Modul pro správu skladu

Modul pro správu skladu spolu s *modulem pro správu zakázek* tvoří stěžejní část celé desktopové aplikace. Funkcionalita *modulu pro správu zakázek* již popsána byla. Jaká je tedy funkcionalita *modulu pro správu skladu*? Odpověď na tuto otázku nacházíme v ilustraci 3-6, kde jsem pomocí use case diagramu celou situaci znázornil.



Obrázek 3-6: Diagram případů užití znázorňující funkcionalitu *modulu pro správu skladu*

Modul pro správu skladu bude zobrazovat příjemky a výdejky materiálů spolu se seznamem, na kterém budou zobrazeny jednotlivé položky materiálu korespondující se zvolenou příjemkou nebo výdejkou. V tomto modulu bude mít dále uživatel možnost vytvořit novou příjemku/výdejku materiálu, modifikovat zvolenou příjemku/výdejku materiálu, odstranit zvolenou příjemku/výdejku materiálu, exportovat zvolenou příjemku/výdejku materiálu do souboru a zobrazit stav skladu. Možnost zobrazení stavu skladu vyvolá dialog, ve kterém bude mít uživatel k dispozici dvě možnosti – možnost filtrování skladu a možnost exportování stavu skladu do formátů *.xls*, *.xml* nebo *.csv*.

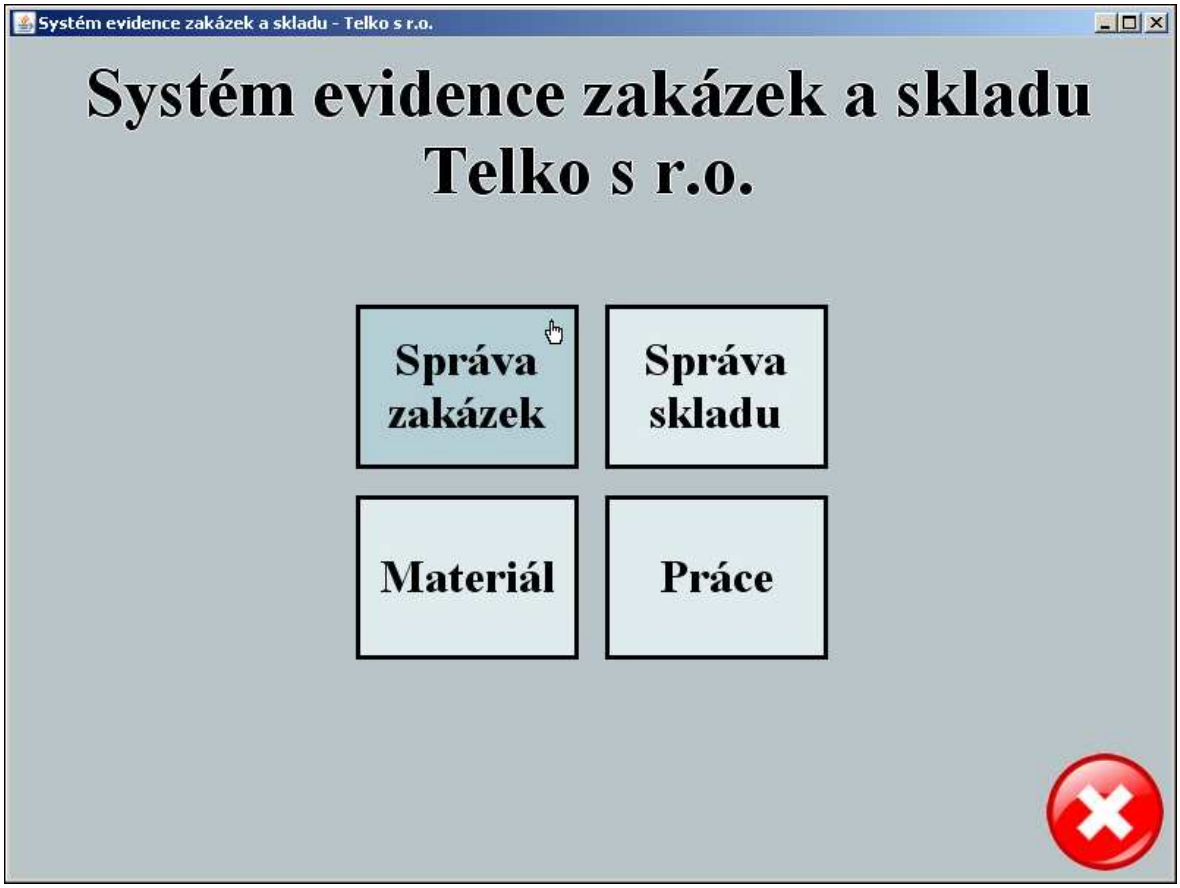
3.4.4 Modul pro správu položek materiálu a práce

Zbylé dva moduly, *modul pro správu položek materiálu* a *modul pro správu položek prací*, o kterých se jen stručně zmíním v této krátké kapitole, budou plnit funkce administrativního charakteru. Kromě přidávání nových položek, modifikace, odstraňování a oceňování stávajících položek, bude v těchto modulech moci uživatel vytvářet své vlastní skupiny tak, jak sám uzná za vhodné. V oficiálním seznamu (ceníku), který jsem měl k dispozici, existují stovky položek, přičemž zákazník v současné době využívá pouhý zlomek z celkového počtu. Tím, že si uživatel vytvoří své vlastní skupiny a přiřadí k nim své oblíbené položky, ušetří v budoucnu značné množství času.

3.5 Návrh grafického uživatelského rozhraní aplikace

Grafické uživatelské rozhraní, nebo chcete-li GUI (graphical user interface), je typ uživatelského rozhraní umožňující grafickou interakci člověka a počítače. Tato interakce je možná díky existenci grafických ovládacích komponent v aplikaci (např. tlačítko, zaškrtávací políčko, seznam apod.). Aplikace, kterou mám v plánu naprogramovat, bude obsahovat celou řadu grafických ovládacích komponent. Tuto část diplomové práce zaměřím na návrh GUI vybraných komponent. Dříve než začnu se samotným výkladem, chtěl bych ještě uvést, že aplikace bude optimalizována pro minimální rozměr 800x600 pixelů. Maximální rozměr aplikace nebude limitován. Minimální rozměr jsem stanovil zejména z důvodu možnosti využití aplikace na dnes velmi moderních a populárních zařízeních typu netbook. Volba menšího rozměru aplikace vývojáře sice limituje, ale zároveň ho nutí efektivně využívat prostor, který má k dispozici.

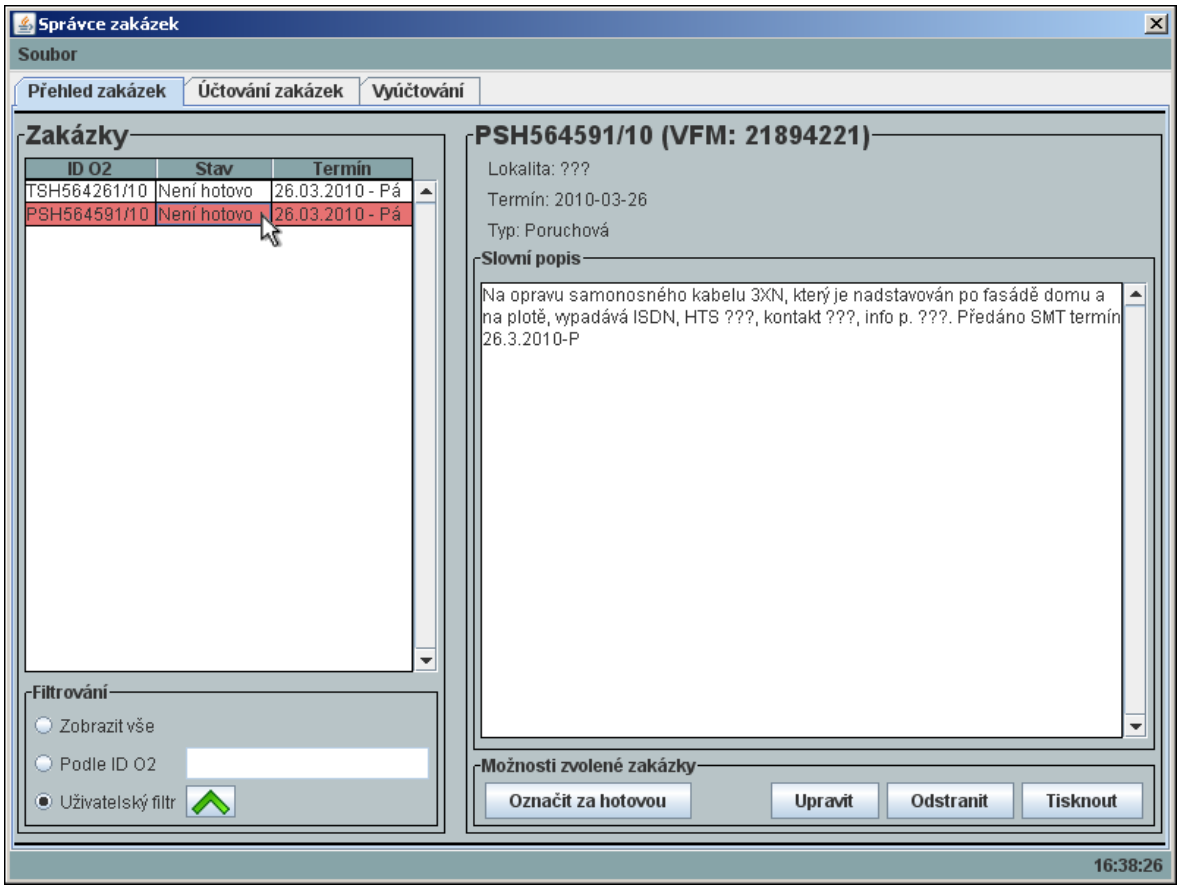
První komponentou, o které se zmíním, je *Hlavní okno*. Jedná se o zastřešující kontejner, přes který mohou být spouštěny další části aplikace. Po startu programu bude *Hlavní okno* tím prvním, co uživatel uvidí. Běh aplikace je determinován touto komponentou. Pokud komponentu vypneme, vypneme zároveň celou aplikaci. Grafický návrh komponenty *Hlavního okna* je znázorněn na obrázku 3-7.



Obrázek 3-7: Návrh GUI *Hlavního okna*

Na obrázku 3-7 vidíme čtyři tlačítka v centrální části (určená pro spouštění konkrétních modulů) a jedno tlačítko v pravé dolní části (určené k ukončení aplikace). Všechna tlačítka znázorněná na obrázku změní po najetí kurzoru myši svůj vzhled tak, aby uživatel poznal, že slouží k interakci s programem. Za zmínku ještě stojí napsat, že v levé dolní části obrázku 3-7 zatím chybí ovládací komponenta pro import a export databáze, která by dle funkcionality tohoto modulu (viz *zastřešující modul*) měla být v budoucnu implementována.

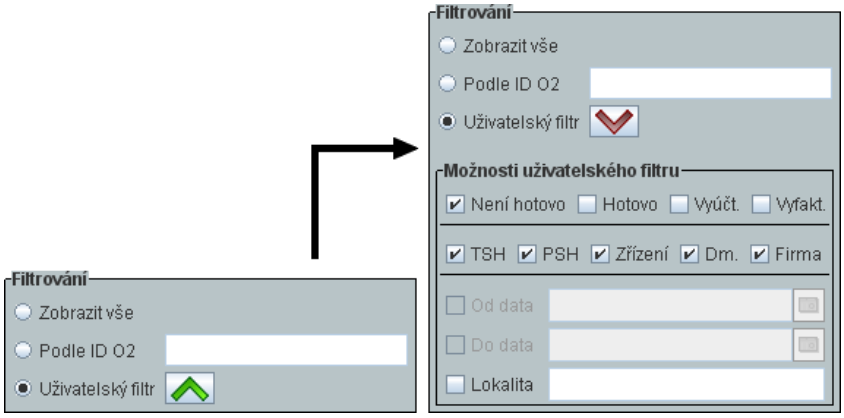
Další vybranou komponentou je *Správce zakázek*. Komponenta je složena ze tří hlavních panelů, které se dají přepínat Jedná se o panely *Přehled zakázek*, *Účtování zakázek* a *Vyúčtování*. Na obrázku 3-8 je zachycen panel *Přehled zakázek*.



Obrázek 3-8: Návrh GUI *Správce zakázek* (panel *Přehled zakázek*)

V levé části panelu *Přehled zakázek* je situována tabulka obsahující dosud nerealizované zakázky (při použití výchozího nastavení filtrování). V pravé části lze spatřit detaily uživatelem zvolené zakázky. V pravé dolní části, má uživatel na výběr několik možností. První z nich je označení zakázky za hotovou. Tuto možnost má uživatel k dispozici pouze tehdy, když se zakázka nachází ve stavu *Není hotovo* (doposud nebyla realizována). Po kliknutí na tlačítko *Označit za hotovou* se stav zakázky nastaví na hodnotu *Hotovo* a při použití implicitního uživatelského filtru zmizne vybraná zakázka ze seznamu, který je umístěn na tomto panelu (zobrazí se v seznamu na panelu *Účtování zakázek*). Dále má uživatel k dispozici možnosti *Upravit*, *Odstranit* a *Tisknout*. Přidat novou zakázku lze

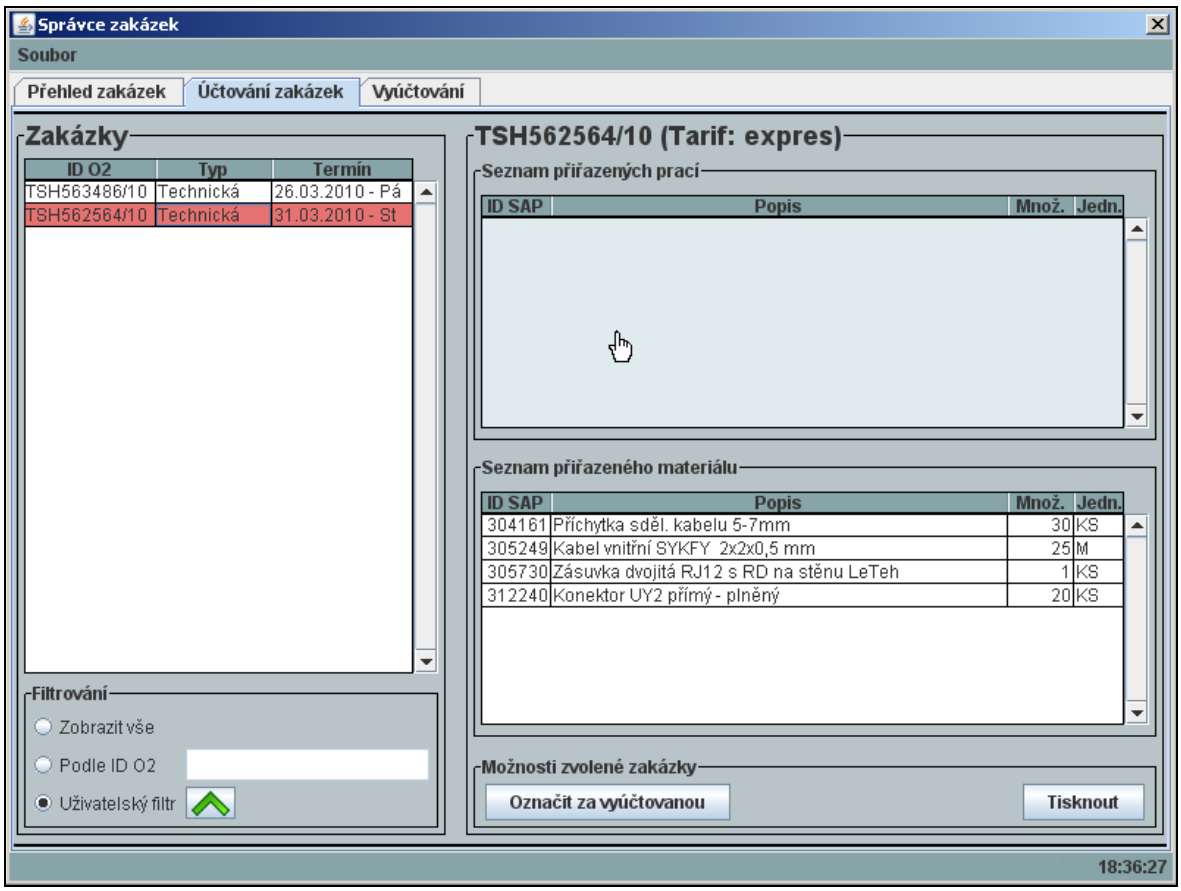
prostřednictvím možnosti *Soubor/Nová zakázka* v *Menu* nebo stisknutím klávesy „+“ na numerické klávesnici. V levé dolní části panelu se nachází komponenta pro filtrování zakázek. Na filtru je implicitně nastavena možnost *Uživatelský filtr*. Pokud se uživatel rozhodne upravit *Uživatelský filtr*, klikne na tlačítko se symbolem zelené šipky (stříšky) a zobrazí se mu tak další filtrovací kritéria, jak ilustruje obrázek 3-9. Poté, co si uživatel navolil svá vlastní kritéria pro filtrování zakázek, může komponentu pro filtrování znovu zminimalizovat a to tak, že klikne na tlačítko červené šipky (obrácené stříšky).



Obrázek 3-9: Rozšířené možnosti komponenty *Filtrování*

Poslední komponentou, na které budu demonstrovat grafické uživatelské rozhraní aplikace, je panel *Účtování zakázek*. Tento panel se nachází ve *Správci zakázek* a jeho podobu zachycuje ilustrace 3.10 uvedená na následující straně. Levá část panelu *Účtování zakázek* je téměř totožná s panelem *Přehled zakázek*. Rozdíl tvoří pouze položky uvedené na seznamu zakázek. Při použití defaultního filtru se na tomto seznamu zobrazují pouze položky (zakázky), které již byly realizovány a zároveň ještě nebyly označeny jako *Vyúčtované*. Na seznamu jsou tedy uvedeny pouze ty zakázky, které mají stav nastaven na hodnotu *Hotovo*. V pravé části můžeme vidět další dva seznamy. První seznam reprezentuje výčet provedených prací vybrané zakázky. Druhý seznam uvádí položky materiálu spotřebovaného při realizaci zakázky. Pokud uživatel bude chtít modifikovat (přidat nebo odebrat) položky některého z výše zmíněných seznamů, jednoduše najede myší na seznam, který posléze lehce zmodrá a klikne na něj levým tlačítkem myši. Poté, co tak uživatel učiní, mu bude zobrazen nový dialog, ve kterém vybere konkrétní položky a přiřadí je do seznamu. Když uživatel k zakázce přiřadí spotřebovaný materiál a vykonané

práce, může použít možnost *Označit za vyúčtovanou*. (možnost je přístupná i v případě, že jsou oba seznamy prázdné). Jakmile tak uživatel učiní, zakázka změní svůj stav na hodnotu *Vyúčtované* a ze seznamu zakázek uvedeném na panelu Účtování zakázek zmizí.



Obrázek 3-10: Návrh GUI panelu *Účtování zakázek*

Popisem grafického uživatelského rozhraní jednotlivých komponent aplikace bych mohl pokračovat i nadále. Já si však myslím, že demonstrativních příkladů už bylo uvedeno více než dost. Proto nyní změním téma a svou pozornost budu věnovat možnostem portability aplikace.

3.6 Portabilita aplikace

K zajištění portability aplikace, je nezbytné disponovat fungujícím databázovým serverem a platformově nezávislým programem. Příkladem platformově nezávislé aplikace, je např. aplikace napsaná v programovacím jazyce Java. Budeme-li uvažovat o aplikaci napsané v tomto programovacím jazyce, budeme potřebovat ještě jednu věc – JRE (Java Runtime Enviroment), zajišťující běh Java aplikací napříč různými platformami.

Kdybychom realizovali databázový server tak, že bychom k němu mohli přistupovat přes internet, ušetřili bychom si tím z hlediska portability aplikace nemalé množství komplikací. Má aplikace je v současné době navržena pouze pro jednoho uživatele a nemusím se tudíž obávat vícenásobného přístupu k datům, jako by tomu bylo u aplikace pro více uživatelů. Tím, že bych databázový server umístil na internet, bych aplikaci vystavil řadě hrozeb. Z tohoto důvodu jsem se rozhodl distribuovat databázový server spolu s aplikací. Při výběru databázového serveru jsem kladl nároky zejména na nenáročnost serveru (aby co nejméně zatěžoval hostitelský počítač). Po krátkých úvahách jsem zvolil databázový server MySQL.

Vytvořená aplikace bude spolu s instalačním souborem serveru MySQL, instalačním souborem JRE, třemi posledními zálohami serveru MySQL a manuálem k instalaci a spuštění uložena na flash disku. Uživatel by si pro případ ztráty nebo poškození flash disku mohl pořídit jednu nebo i více kopií celého obsahu flash disku. Aby vše fungovalo tak jak má, uživatel bude aktivně používat pouze jednu kopii celé distribuce. Pokud uživatel bude chtít aplikaci spustit v novém prostředí, nainstaluje z flash disku JRE a server MySQL, který následně podle manuálu nakonfiguruje. Poté spustí aplikaci a v *Hlavním okně* zvolí možnost pro synchronizaci databáze. Touto cestou se z aktuálního zálohového souboru databáze MySQL uloženého na flash disku obnoví databázový server lokalizovaný v novém prostředí. Uživatel poté může pracovat s aktuálními daty. Pokud uživatel ukončí řádně svou činnost, provede se záloha databáze na flash disk (nejstarší soubor je nahrazen souborem nejnovějším tak, aby na flash disku byly vždy 3 soubory se zálohami databáze). Pokud uživatel přejde do prostředí, kde již měl aplikaci funkční, vynechá instalační a konfigurační kroky MySQL a JRE a jednoduše program spustí a provede synchronizaci lokálního databázového serveru. Pro zvýšení bezpečnosti doporučuji celý flash disk uzamknout (nastavit bezpečné heslo).

4. Implementace databázové aplikace v programovacím jazyce Java

Třetí kapitola diplomové práce bude věnována implementaci databázové aplikace. Kapitolu jsem rozdělil do dvou hlavních částí. V první části se zaměřím na stručný popis použitých technologií – uvedu zde informace o databázovém serveru MySQL, technologii Java a jejích knihovnách javax.swing.* a JExcelApi. Ve druhé části pak přejdu k implementaci vybraných komponent.

4.1 Použité technologie

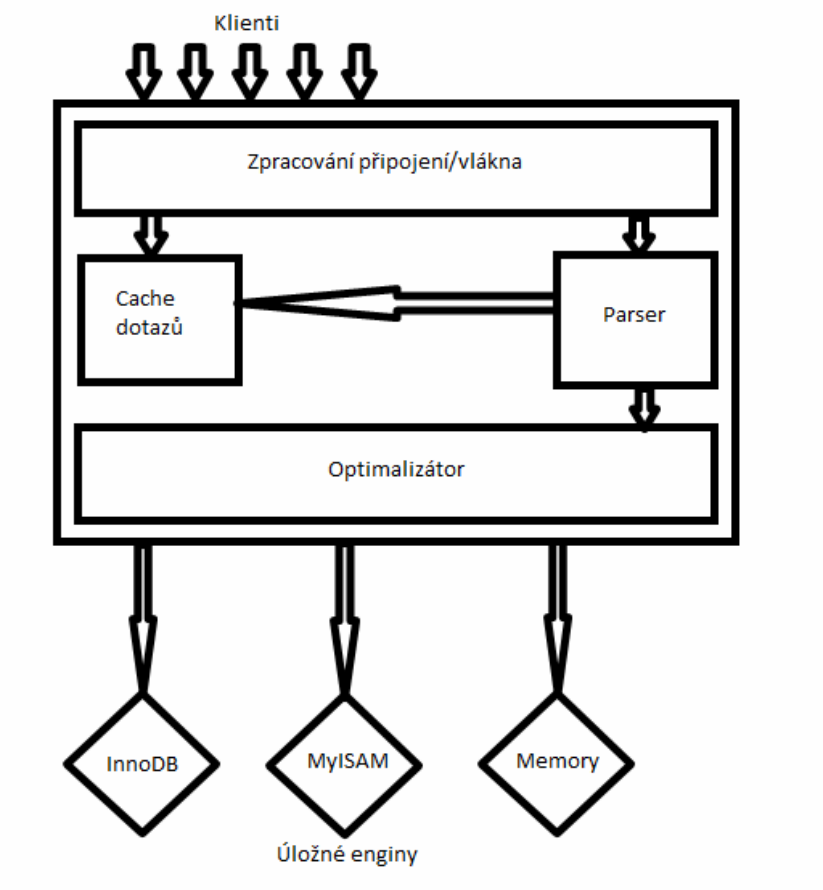
V této části diplomové nejprve stručně popíši databázový server MySQL, technologii Java, knihovnu javax.swing.*, která je součástí rozšíření standardních knihoven jazyka Java (viz balíček javax.*) a nakonec knihovnu JExcelApi s jejíž pomocí bude aplikace provádět import nebo export dat z nebo do formátu *.xls*.

4.1.1 Databázový server MySQL

Relační databázový server MySQL přišel na svět původně jako interní firemní projekt, v jehož čele stáli zaměstnanci Michael Widenius a David Axmark. První vydání pro veřejnost bylo realizováno v roce 1995 softwarovou firmou TCX DataKonsult AB se sídlem ve Švédsku v Uppsale. Tento software se stal brzy velmi populárním, jeho tvůrci si začali uvědomovat jeho potenciál a proto založili MySQL AB, což je firma obstarávající veškeré záležitosti okolo MySQL, služby specifické pro MySQL a nabídky produktů. V současnosti je MySQL vlastněno společností Sun Microsystems, dceřinou společností Oracle Corporation. [4]

MySQL je multiplatformní databáze a komunikace s ní probíhá pomocí jazyka SQL (Structured Query Language). Podobně jako u ostatních SQL databází se jedná o dialekt jazyka SQL spolu s dalšími rozšířeními. Pro svou snadnou implementovatelnost (lze jej instalovat na Linux, MS Windows, ale i další operační systémy), výkon a fakt, že se jedná o volně šiřitelný software, zabírá v současnosti databázový server MySQL poměrně vysoký podíl na trhu používaných databázových systémů. [3i]

MySQL bylo od počátku optimalizováno především pro rychlost, a to i za cenu některých zjednodušení a až donedávna nepodporovalo pohledy, triggery, a uložené procedury. Tyto vlastnosti jsou doplňovány teprve v posledních letech. Architektura MySQL, znázorněná na obrázku 4-1, má široký záběr a je užitečná pro řešení mnoha různorodých úloh.



Obrázek 4-1: Architektura databázového serveru MySQL [3i]

Vrstva, která je úplně nahoře, obsahuje služby, jež nejsou jedinečné pro MySQL. Obsluhují většinu potřebných nástrojů klient/server, které jsou založeny na síti. Ve druhé vrstvě se nachází řídící centrála MySQL, včetně kódu pro rozbor (parsing), analýzu, optimalizaci a pro všechny zabudované funkce. Na této úrovni se nachází veškerá funkcionalita, která je poskytována prostřednictvím úložných enginů. Třetí vrstva obsahuje úložné enginy. Ty mají na starosti ukládání a získávání všech dat uložených v MySQL. Server komunikuje s úložnými enginy prostřednictvím API úložných enginů. Toto rozhraní

skrývá rozdíly mezi jednotlivými úložnými enginy a činí je na vrstvě dotazů velmi transparentními. API obsahuje několik desítek nízkoúrovňových funkcí, které provádějí operace jako „zahájit transakci“ nebo „získat řádek, který má tento primární klíč“. Úložné enginy nedělají rozbor SQL a nekomunikují mezi sebou – jednoduše pouze odpovídají na požadavky serveru. [3i]

Správa připojení a bezpečnost – každé klientské připojení dostane uvnitř serverového procesu vlastní vlákno (thread). Dotazy tohoto připojení se vykonávají uvnitř tohoto jediného vlákna, které zase sídlí na jednom jádru nebo CPU. Protože server udržuje vlákna v cache, nemusejí se vytvářet a likvidovat pro každé nové připojení. Autentizace je založena na uživatelském jménu, hostiteli (odkud pocházejí) a heslu. Dají se také používat certifikáty X509 přes připojení SSL. Jakmile se klient připojí, server pro každý dotaz vydaný klientem ověřuje, zda-li má patřičná oprávnění pro akci, kterou chce vykonat. [3i]

Optimalizace a vykonávání – MySQL provádí rozbor dotazů proto, aby vytvořil interní stromovou strukturu (parse tree), pak aplikuje několik optimalizací. Může dotaz přepsat, určit pořadí, v němž bude číst tabulky, zvolit, které indexy použije atd. Prostřednictvím speciálních klíčových slov může programátor optimalizátoru předat tzv. pokyny (hints), jimiž se dá ovlivnit rozhodovací proces. Optimalizátor se ve skutečnosti nestará o to, který úložný engine používá konkrétní tabulka. Úložný engine ovšem ovlivňuje, jak server optimalizuje dotaz. Optimalizátor od úložného enginu zjišťuje, zdali má jistou výbavu, dotazuje se na náklady určitých operací a na statistiky o datech tabulky. Než server začne s rozborem dotazu, obrátí se nejprve na cache dotazů (query cache), kam může ukládat pouze příkazy SELECT společně s jejich výslednými sadami. Jestliže někdo vydá dotaz, který je identický s nějakým dotazem, který je už k dispozici v cache, server nemusí dělat vůbec žádný rozbor, nemusí nic optimalizovat a dokonce nemusí dotaz ani vykonat – jednoduše pouze předá zpět uloženou výslednou sadu. [3i]

Úložné enginy (úložiště dat) – MySQL ukládá každou databázi (v případě MySQL se jedná o schéma) do podadresáře svého datového adresáře na podkladovém souborovém systému. Když vytvoříte nějakou tabulku, MySQL uloží definici tabulky do souboru *.frm*, který má název shodný s názvem tabulky. Pokud tedy vytvoříte tabulku nazvanou MyTable, MySQL vytvoří definici tabulky jako soubor *MyTable.frm*. Protože MySQL používá při ukládání definic souborový systém, otázka rozlišování velikosti písmen je závislá na platformě. Pokud databázi MySQL nainstalujeme na operační systém Windows,

velikost písmen v názvech tabulek a databází nebude rozlišována, na unixovýxh systémech se velikost písmen rozlišuje. Každý úložný engine ukládá tabulky a indexy jinak, definici tabulky ovšem zpracovává samotný server. MySQL nabízí několik typů úložných enginů (storage engine), které se liší svými možnostmi, použitím a způsobem ukládání dat do souborů. Mezi tyto enginy patří např. MyISAM, InnoDB, BerkeleyDB, CSV, MERGE, Falcon atd. Úložné enginy fungují jako moduly, které lze k distribuci tohoto databázového systému doinstalovat. Jejich aktuální seznam lze zjistit příkazem *SHOW ENGINES*. [3i]

Kódování a znakové sady – od verze 4.1 MySQL řeší ukládání řetězců s podporou Unicode pomocí nastavení znakové sady a collation. To představuje souhrn způsobů, jak k takto uloženému textu přistupovat – porovnávání (s ohledem na případné národnostní zvyklosti), řazení, citlivost velkých malých písmen, ligatur, transkripcí speciálních znaků apod. Znaková sada a collation mohou být nastaveny individuálně pro daný (textový) sloupec, mimo je možnost nastavit defaultní sadu a collation pro tabulku (zdědí je vytvářené sloupce, u kterých nebyla explicitně vybrána), i celá databáze (tu zase kaskádově zdědí v ní vytvářené tabulky, pokud pro ně není výslovně nastavena). I jednotlivé collations jsou modulární a existují v podobě textových souborů. Jejich aktuální výčet lze zjistit příkazem *SHOW COLLATION*. [3i]

4.1.2 Java

Začnu nejprve trochou historie. Od roku 1991 vyvíjela firma Sun Microsystems programovací jazyk na principech C a C++ pro vestavěné systémy, což je odborný termín pro běžná elektronická zařízení ovládaná zabudovaným mikroprocesorem. Jazyk měl původně název Oak (dub), podle dubu, který stál před oknem pana Goslinga, vedoucího týmu. Posléze se zjistilo, že již programovací jazyk s tímto jménem existuje, takže vývojová skupina hledala název nový a po návštěvě firemního bufetu padl návrh Java, což znamená v americkém slangu „kafe“. Projektu se příliš nedařilo, ale v roce 1993 si firma Sun uvědomila vzrůstající důležitost webových aplikací a možnosti využít Javu pro jejich programování. V květnu roku 1995 byla Java firmou Sun oficiálně představena na konferenci. Již během této přednášky začalo být mnohým účastníkům zřejmé, že se jedná o programovací jazyk, který bude hrát významnou úlohu při programování jak „běžných“, tak zejména i webových aplikací. [5]

Java je čistě softwarová platforma, která běží nad různými hardwarově založenými platformami. Díky této vlastnosti Java platforma umožňuje vývoj a multiplatformní nasazení jak desktopových, tak i serverových aplikací. Součástí Javy jsou také třídy podporující vývoj webových služeb a třídy poskytující základ pro vývoj enterprise aplikací. Java platforma je složena ze tří hlavních částí.

- **Programovací jazyk Java** je syntakticky podobný jazyku C++, liší se však v provedení. Zatímco u programovacího jazyka C++ jsou programátoři zodpovědní za přidělování a uvolňování paměti, v programovacím jazyce Java tato nutnost odpadá. Dalším rozdílem mezi jazykem Java a C++ je fakt, že v programovacím jazyce Java není možná vícenásobná dědičnost (možný zdroj zmatků). Nemožnost dědit od více než jednoho předka je v Javě kompenzována možností implementovat libovolný počet rozhraní.
- **Virtuální stroj Javy (Java Virtual Machine, JVM)** tvoří základ platformy Java. JVM interpretuje bajkód (zkompilovaný Java program) a zajišťuje běh aplikace na různých platformách. Tvoří prostředníka mezi hardwarovou platformou a programem napsaným v jazyce Java.
- **Java API (Application Programming Interface)** představuje poslední pilíř Java platformy. V podstatě se jedná o souhrn knihoven tříd, které jsou považovány za standardní a musejí se vyskytovat v každém prostředí, kde chceme program napsaný v programovacím jazyce Java použít. Díky těmto standardním knihovnám zabírají námi napsané a zkompilované aplikace v Javě mnohem méně místa na disku. Pokud požadovaná knihovna není ve standardním API obsažena, jednoduše se jí do projektu importujeme.

V současné době se můžeme setkat s trojím členěním Java platforem. **Java SE (Standard Edition)** je platforma navržená pro stolní počítače. Nejčastěji běží nad operačními systémy Sun Solaris, Mac OS X, Linux nebo Microsoft Windows. **Java EE (Enterprise Edition)** je komplexní platforma určená pro víceuživatelské aplikace zaměřené na široké použití. Java EE vychází z platformy Java SE rozšířené o rozhraní pro podporu webových aplikací běžících na straně serveru. Poslední platforma, **Java ME (Micro Edition)**, je sada technologií a specifikací vyvinutá pro malá zařízení typu pager nebo mobilní telefon. Java ME využívá odlehčené Java SE API a definuje řadu specifických rozhraní. [4i]

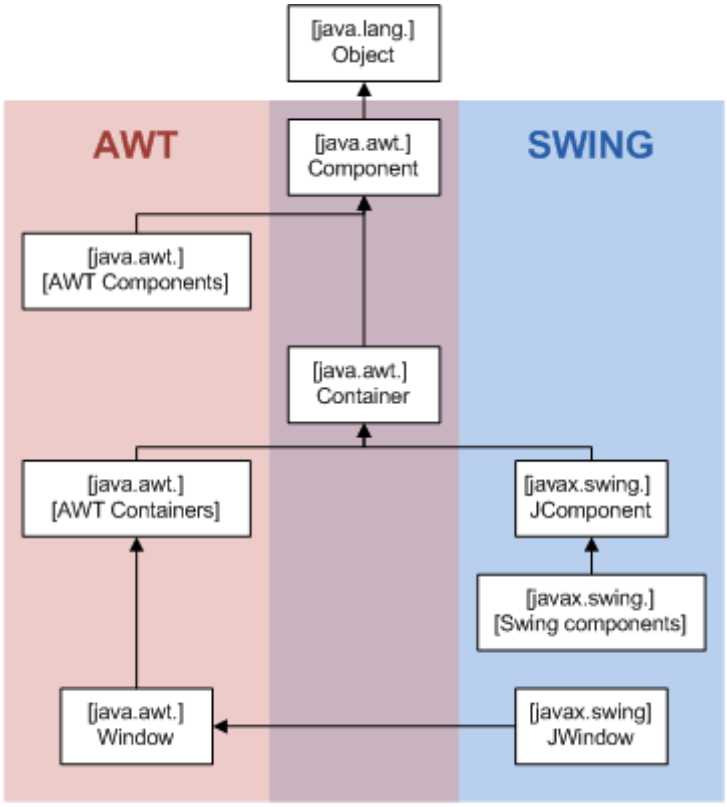
4.1.3 Java – knihovna Swing

Swing je knihovna uživatelských prvků na platformě Java určená pro ovládání počítače pomocí grafického rozhraní. Knihovna Swing poskytuje aplikační rozhraní pro tvorbu a obsluhu klasického grafického uživatelského rozhraní. Pomocí Swingu je možno vytvářet okna, dialogy, tlačítka, rámečky, rozbalovací seznamy, atd. [7i]

Původní nástroj pro tvorbu grafického uživatelského rozhraní je Abstract Window Toolkit (AWT), vyvíjený firmou Sun Microsystems jako součást Javy. První verze vyšla v roce 1995, spolu s první verzí jazyka Java. Během dalšího vývoje se ale u AWT projevily chyby v návrhu a další problémy. Např. bylo závislé na platformě a tím porušovalo jeden z principů, na kterém je jazyk Java postaven. V roce 1997 Sun upustil od dalšího vývoje AWT a začal vyvíjet Swing. Základem byly Internet Foundation Classes (IFC), vyvinuté Netscape Communications Corporation na konci roku 1996, jako nástroj pro tvorbu grafického uživatelského rozhraní pro Javu. V roce 1997 došlo ke spojení IFC a dalších technologií od Netscape a Sun Microsystems do Java Foundation Classes (JFC), jehož je Swing nyní součástí. Swing je vyvíjen jako kombinace AWT, IFC a dalších technologií. Swing je nedílnou součástí Java SE od verze 1.2. Do té doby byl dostupný pouze jako knihovna k samostatnému stažení. [7i]

Hierarchie tříd grafických komponent Swingu je založena na rodičích z AWT, jak ilustruje obrázek 4-2 na následující straně. Stejně jako každá třída Javy, i třídy grafického uživatelského rozhraní jsou potomky třídy java.lang.Object. Základním stavebním kamenem všech grafických komponent je třída java.awt.Component. Instance třídy java.awt.Component představují objekty, které mají svoji grafickou reprezentaci a mohou být zobrazeny na monitoru uživatele. Všechno, co Java na monitoru vykreslí, je instance třídy java.awt.Component. Různé ovládací prvky (tlačítko, zaškrtávací políčko, seznam aj.) v AWT jsou definovány jako potomci třídy java.awt.Component. Od třídy java.awt.Component je dále odvozena třída java.awt.Container (kontejner). Kontejner je specifická grafická komponenta, která v sobě může držet a vykreslovat ostatní grafické komponenty. Jakým způsobem se komponenty v kontejneru nakreslí, určuje tzv. správce rozložení (implementace rozhraní java.awt.LayoutManager). V AWT jsou od třídy java.awt.Container odvozeny různé prvky, např. třída java.awt.Window nebo java.awt.Panel. Základním kamenem knihovny Swing je třída javax.swing.JComponent. Tato třída je rodičem každé swingovské komponenty. Jak je patrno z obrázku 4-2,

javax.swing.JComponent je potomkem java.awt.Container. Tudíž všechny swingovské prvky v sobě mohou držet další komponenty. Potomci třídy javax.swing.JComponent jsou samotné swingovské komponenty, jako je javax.swing.JButton (tlačítko), javax.swing.JList (seznam), javax.swing.JPanel (obecný panel) apod. Všechny swingovské komponenty mají před svým logickým názvem písmeno „J“, aby byly rozlišitelné od svých protějšků z AWT. K této obecné hierarchii existují výjimky. Tyto výjimky jsou představovány okny a obecně všemi nejvyššími kontejnery (top-level containers – okna, dialogy, applety). Okna jsou obecně potomci třídy java.awt.Window, a nemohou proto být potomci třídy javax.swing.JComponent (v Javě nelze dědit od více tříd). Swingovské protějšky k java.awt.Window a jeho potomků (java.awt.Frame, java.awt.Dialog, atd.) jsou definovány jako potomci těchto tříd s „J“ na začátku. Tj. hlavní swingovské okno javax.swing.JFrame je definováno jako potomek java.awt.Frame. [7i]



Obrázek 4-2: Hierarchie tříd grafických komponent Swingu založená na rodičích z AWT [7i]

AWT bylo koncipováno jako rozhraní mezi Javou a grafickým API platformy. To znamená, že všechny grafické komponenty byly vykreslovány samotným systémem a měly tedy nativní podobu. Např. JVM na Windows využívala Win32 API (konkrétně knihovny gdi32.dll a další) ke kreslení AWT komponent. Tyto nativně vykreslené prvky se nazývají „těžké“ (heavyweight), protože měly své vlastní nativní neprůhledné okno v systému. Na systém se spoléhaly i v dalších věcech, např. zajišťování pozice na ose Z (která komponenta je v popředí a která v pozadí). Swing přináší jiný přístup. Každá komponenta je sama zodpovědná za svůj vlastní vzhled a nespoléhá se na systém, sama definuje, jak má vypadat, a na požádání se vykreslí na zobrazovací zařízení. Proces vykreslení se odehrává přímo v Javě. Operačnímu systému se pouze předá hotový obrázek, který je třeba namalovat. Tomuto přístupu se říká „lehký“ (lightweight). To, jak komponenty budou vypadat, již tedy nezáleží na operačním systému. Swing navíc přidává podporu pro dvojitou vyrovnávací paměť (double buffering), která umožňuje plynulé kreslení grafických prvků. Dále přidává podporu průhlednosti a částečné průhlednosti, podporu pro optimalizované kreslení v případě překrývajících se komponent a další technologie. Pro kreslení využívá další část JFC, a to Java 2D API. Kreslení nejvyšších kontejnerů (oken, dialogů apod.) je ve Swingu převzato z AWT, tudíž okna jako taková jsou platformě závislá (nelze mít okno na platformě, která okna nepodporuje). Vzhled oken však je konfigurovatelný. [7i]

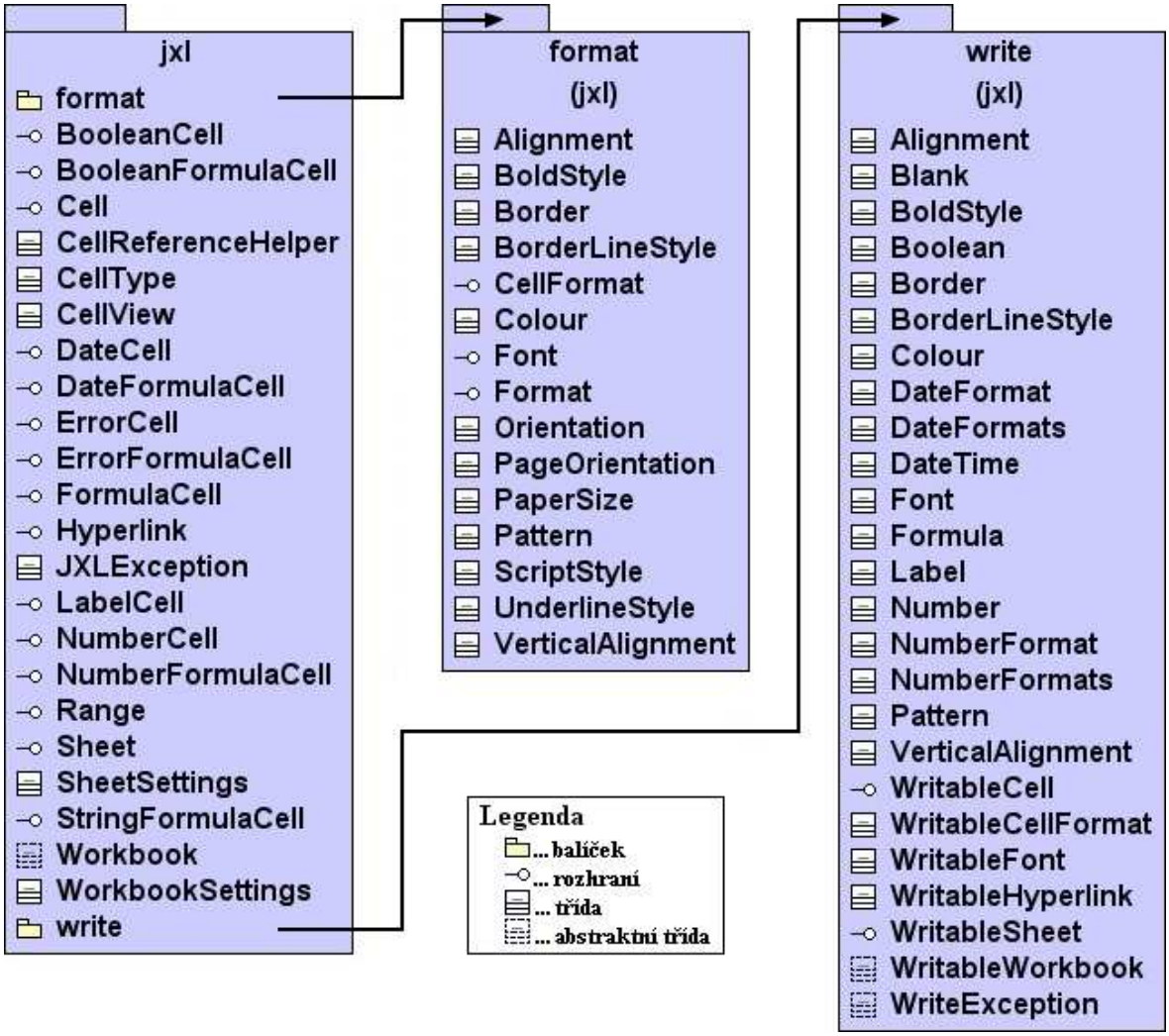
4.1.4 Java – knihovna JExcelApi

Knihovna JExcelAPI, nebo-li Java Excel API, je open source Java API umožňující dynamické načítání, zapisování nebo modifikaci dat spojených s formátem *.xls*. Pomocí JExcelAPI můžeme tedy například načíst data z Excelu, pozměnit je a poté zapsat do libovolného typu souboru (*.txt*, *.xls*, *.xml*, aj.) nebo do databáze. Protože se jedná o ryzí Java knihovnu, je knihovna JExcelAPI vhodná pro vývoj multiplatformních aplikaci. V následujících odrážkách uvádím výčet několika specifik knihovny JExcelAPI:

- čtení údajů ze sešitů (Excel verze 95, 97, 2000, XP, 2003),
- čtení a zápis vzorců (Excel verze 97 nebo novější),
- generování tabulek ve formátu Excel 2000,
- podpora textových, číselných a datových formátů,

- podpora stínování, ohraničování a vybarvování buněk,
- modifikace existujících sešitů,
- je mezinárodní - umožňuje nastavení země, jazyka, znakové sady,
- podpora kopírování grafů,
- podpora vkládání a kopírování obrázků do tabulek atd.

Knihovna JExcelAPI není nijak rozsáhlá. Z toho důvodu je použití knihovny JExcelAPI poměrně jednoduché a intuitivní (dokumentace knihovny je velmi stručná). Výčet jednotlivých elementů knihovny JExcelAPI uvádím na obrázku 4-3. [5i]



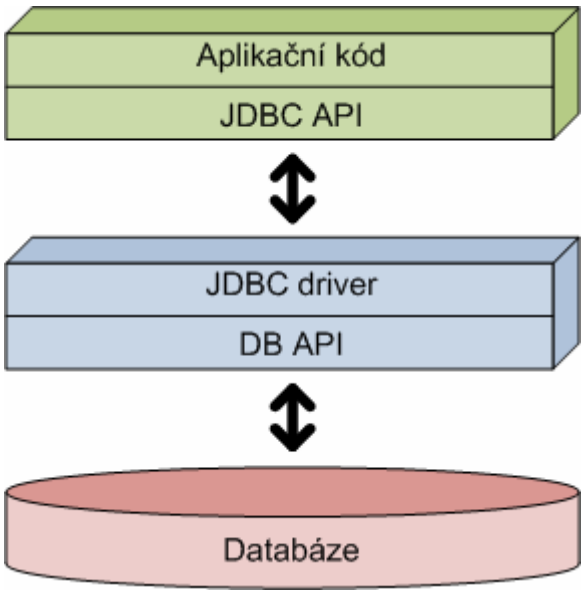
Obrázek 4-3: Přehled jednotlivých elementů knihovny JExcelAPI [5i]

4.2 Implementace vybraných komponent

V této části proberu implementaci některých komponent, které budou součástí aplikace evidenčního systému firmy Telko s r.o. Samozřejmě se nebudu pokoušet popsat implementaci všech komponent, protože si myslím, že by to přesahovalo rozměry mé diplomové práce. Postupně se tedy zaměřím na implementaci navázání konektivity s databázovým serverem MySQL, na implementaci komponenty javax.swing.JTable a nakonec na naplnění databázového serveru MySQL požadovanými vstupními daty.

4.2.1 Navázání konektivity s databázovým serverem MySQL

Abychom mohli v Java aplikaci používat dotazy SQL, potřebujeme nejprve získat objekt jazyka Java, s jehož pomocí budeme schopni komunikovat s databázovým serverem. Objekt, o kterém jsem teď psal, je instancí rozhraní java.sql.Connection. Než ale získáme instanci rozhraní java.sql.Connection, musíme mít nejprve k dispozici správný JDBC driver, který je obvykle poskytován výrobcem databázového serveru a slouží jako prostředník mezi aplikací a databázovým serverem. Pro komunikaci se samotným JDBC driverem se využívá JDBC API, které je realizováno sadou rozhraní a tříd napsaných v programovacím jazyce Java. Celou situaci ilustruje obrázek 4-4. [8]



Obrázek 4-4: Využití JDBC driveru ke komunikaci mezi aplikací a databázovým serverem [8]

Já jsem pro svou aplikaci využil databázový server MySQL. Z toho důvodu potřebuji mít k dispozici JDBC driver, který je kompatibilní s databázovým serverem MySQL. Z oficiálních webových stránek produktu MySQL jsem si tedy stáhl potřebný ovladač (JDBC Driver for MySQL) a importoval jsem jej do svého projektu. Dalším krokem je zaregistrování ovladače. To lze v Javě provést dvěma způsoby. Buď příkazem Class.forName("String className") nebo new com.mysql.jdbc.Driver(). Poté, co jsem zaregistroval JDBC driver, již mohu získat dříve zmiňovaný objekt typu java.sql.Connection. Ten získám prostřednictvím objektu java.sql.DriverManager a jeho metody java.sql.Connection getConnection(String url, String user, String password). Získání objektu java.sql.Connection je v programu realizováno ve třídě AStarter (potomek třídy javax.swing.JFrame), která zastřešuje celou aplikaci. Implementace navázání konektivity je pomocí programovacího jazyka Java uvedena níže (viz příklad 4-1):

```
public static java.sql.Connection con;
...
try {
        Class.forName("com.mysql.jdbc.Driver");
        con = java.sql.DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/xxxxxxxx", "xxxxxxxx", "xxxxxxxx");
        con.setAutoCommit(false);
} catch (java.sql.SQLException ex) {
        javax.swing.JOptionPane.showMessageDialog(this, ex.toString());
} catch (Exception e) {
        javax.swing.JOptionPane.showMessageDialog(this, e.toString());
}
```

Příklad 4-1: Navázání konektivity s MySQL v programovacím jazyze Java

V příkladu 4-1 nejprve deklaruji proměnnou con (typ java.sql.Connection), ke které zatím nepřiřazuji žádnou hodnotu (objekt). Všimněme si, že v deklaraci proměnné je uvedeno klíčové slovo static (objekt con je dostupný bez toho, aniž bychom vytvářeli instanci třídy, která tuto proměnnou obsahuje). Dále v bloku try konstrukce try-catch zaregistruji ovladač MySQL serveru využitím statické metody forName třídy java.lang.Class. Nakonec pomocí statické metody getConnection třídy java.sql.DriverManager inicializuji proměnnou con. Nakonec na proměnné con nastavím vlastnost autoCommit (typ boolean) na hodnotu false (kvůli možnosti provádění transakcí nad databází MySQL).

4.2.2 Implementace komponenty javax.swing.JTable

Komponentu javax.swing.JTable jsem do kapitoly zaměřenou na implementaci vybraných komponent zahrnul zejména pro její četnost výskytů v rámci celé aplikace. Komponenta javax.swing.JTable v podstatě představuje klasickou tabulku, která se používá pro zobrazení výsledků dat, pro hromadné zadávání dat, pro různé početní operace či podobné operace. Data v tabulce mohou, ale nemusí být editovatelná. V tabulce zpravidla rozlišujeme dva typy modelů. Model pro sloupce, který ovlivňuje zobrazení dat ve sloupci či možnosti jejich editace a model reprezentující celou tabulku disponující řadou metod určených pro manipulaci s tabulkou. Komponenta javax.swing.JTable je poměrně komplikovaná a její potenciál je obrovský. [2i]

Pokud chceme vytvořit instanci třídy javax.swing.JTable můžeme využít několika konstruktorů. Já v příkladu 4-2 uvádím pouze tři.

```
javax.swing.JTable tab = new javax.swing.JTable();
                       = new javax.swing.JTable(int rows, int columns);
                       = new javax.swing.JTable(Object[][] data, Object[] colNames);
```

Příklad 4-2: Demonstrace konstruktorů třídy javax.swing.JTable

První z konstruktorů uvedených v příkladu 4-2 vytvoří prázdnou tabulku. Druhý vytvoří prázdnou tabulku s přesně stanoveným počtem řádků (rows) a sloupců (columns). Poslední, třetí konstruktor, vytvoří daty naplněnou tabulku se jmény sloupců. K tomu, aby komponenta javax.swing.JTable mohla být správně zobrazena, je nezbytné ji umístit do kontejneru. Obvykle se komponenta javax.swing.JTable se kombinuje s kontejnerem javax.swing.JScrollPane, jehož využití je takřka nezbytné u rozměrnějších typů tabulek. Kontejner javax.swing.JScrollPane umožňuje posouvání svého obsahu jak vertikálním, tak i horizontálním směrem. Přidání tabulky do kontejneru zachycuje příklad 4-3.

```
javax.swing.JScrollPane sp;
sp = new javax.swing.JScrollPane(tab,sp.VERTICAL_SCROLLBAR_ALWAYS,
        sp.HORIZONTAL_SCROLLBAR_NEVER);
```

Příklad 4-3: Přidání objektu javax.swingJTable do kontejneru javax.swing.JScrollPane

Implicitně nastaveným modelem instance třídy javax.swing.JTable je objekt typu javax.swing.table.DefaultTableModel. Pokud bych například chtěl, aby má tabulka disponovala nějakou novou přidanou funkcionalitou (např. pracovala s daty obsaženými v objektu typu java.util.ArrayList), musel bych vytvořit novou třídu – potomka abstraktní třídy javax.swing.table.AbstractTableModel. Náš potomek musí od svého předka implementovat minimálně tři metody. Jsou jimi public int getRowCount(), public int getColumnCount() a metoda public Object getValueAt(int row, int column). Dále je vhodné přepsat metodu public String getColumnName(int column) pro zobrazení záhlaví tabulky a metodu public Class getColumnClass(int column) pro správné zobrazení objektu v tabulce (např. boolean se zobrazí jako zaškrtávací políčko). Vytvoření potomka třídy javax.swing.table.AbstractTableModel uvádím příkladu 4-4.

```
class PrehledMaterialuTableModel extends javax.swing.table.AbstractTableModel {
	private java.util.ArrayList<MaterialPolozkyEntity> seznamPolozek;

	public PrehledMaterialuTableModel(
		java.util.ArrayList<MaterialPolozkyEntity> seznamPolozek) {
		this.seznamPolozek = seznamPolozek;
	}

	@Override
	public int getRowCount() {
		return seznamPolozek.size();
	}

	@Override
	public int getColumnCount() {
		return 3;
	}

	@Override
	public String getColumnName(int column) {
		switch (column+1) {
			case 1:
				return "ID SAP";
			case 2:
				return "Popis";
			case 3:
				return "Jedn.";
		}
```

```
            return null;
        }

        @Override
        public Class getColumnClass(int columnIndex) {
            return getValueAt(0, columnIndex).getClass();
        }

        @Override
        public Object getValueAt(int row, int column) {
            MaterialPolozkyEntity polozka = seznamPolozek.get(row);
            switch (column+1) {
                case 1:
                    return polozka.getIdSap();
                case 2:
                    return polozka.getPopis();
                case 3:
                    return polozka.getMernaJednotka();
            }
            return null;
        }

        public java.util.ArrayList<MaterialPolozkyEntity> getMaterialPolozky() {
            return seznamPolozek;
        }

        public void setMaterialPolozky(
            java.util.ArrayList<MaterialPolozkyEntity> seznamPolozek) {
            this.seznamPolozek = seznamPolozek;
        }
}
```

Příklad 4-4: Vytvoření potomka třídy javax.swing.table.AbstractTableModel

Z příkladu 4-4 je patrné, že třída PrehledMaterialuTableModel je potomkem abstraktní třídy javax.swing.table.AbstractTableModel. Data jsou pro tento model čerpána z objektu typu java.util.ArrayList<MaterialPolozkyEntity>, tedy parametrizovaného seznamu. Dále lze z kódu vyčíst počet sloupců tabulky (tři) a titulky jednotlivých sloupců („ID SAP“, „Popis“, „Jedn.“). Pokud bychom v aplikaci chtěli vytvořit (použít) instanci třídy javax.swing.JTable, nastavit k ní výše zmíněný model a naplnit jí daty, udělali bychom to například následujícím způsobem (viz příklad 4-5).

```
java.util.ArrayList<MaterialPolozkyEntity> seznamPolozekMaterialu =
        new java.util.ArrayList<MaterialPolozkyEntity>();
...
javax.swing.JTable jTableVsechnyPolozky = new javax.swing.JTable();
jTableVsechnyPolozky.setModel(new PrehledMaterialuTableModel(
        seznamPolozekMaterialu));
jTableVsechnyPolozky.setFillsViewportHeight(true);
jTableVsechnyPolozky.setSelectionBackground(new java.awt.Color(231, 114, 114));
jTableVsechnyPolozky.setSelectionMode(
        javax.swing.ListSelectionModel.SINGLE_SELECTION);
javax.swing.JScrollPane jScrollPane1 = new javax.swing.JScrollPane(
        javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
jScrollPane1.setViewportView(jTableVsechnyPolozky);
...
```

Příklad 4-5: Implementace třídy PrehledMaterialuTableModel

Interpretace výše uvedeného kódu do lidské řeči je zhruba taková. Nejprve se vytvoří parametrizovaný seznam (objekt typu java.util.ArrayList). Poté se naplní daty (tento proces v příkladu zachycen není) z databáze. Dále se vytváří objekt typu javax.swing.JTable a nastavuje se jeho model, který je instancí třídy PrehledMaterialuTableModel. Pak následuje sekvence metod, které postupně specifikují nově vytvářenou tabulku. Nakonec je tabulka umístěna do kontejneru typu javax.swing.JScrollPane.

To, co jsem na předchozích čtyřech stranách popsal, byla pouhá ukázka použití komponenty javax.swing.JTable a zdaleka jsem zde nezahrnul vše, co jsem implementoval do své aplikace.

4.2.3 Naplnění databázového serveru MySQL potřebnými daty

Pro naplnění databáze MySQL vstupními daty (položky práce, položky materiálu, ceny prací, ceny materiálu, testovací zakázky, testovací vyúčtování aj.) jsem vytvořil jednoduchý program v programovacím jazyce Java. Vstupní data jsem od zákazníka obdržel v několika souborech formátu *.xls*. Otázkou však je, jak načíst data z formátu *.xls* do dynamických proměnných programovacího jazyka Java? K tomuto účelu využiji dříve zmíněnou knihovnu JExcelAPI. Implementaci knihovny JExcelAPI demonstruji na příkladu 4-6, který je uveden na další straně.

```
...
int citac = 0;
java.sql.Connection con = null;
java.sql.PreparedStatement pstat = null;
String sqlDotaz = "insert into material_polozky (id_sap, id_sap_stare, popis, " +
        "merna_jednotka, id_kategorie_fk) values (?,?,?,?,?)";
String path = "C:/OneSAP_KZM_S1.xls";

try {
        con = java.sql.DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/xxxxxxxx", "xxxxxxxx", "xxxxxxxx");
        pstat = con.prepareStatement(sqlDotaz);
        jxl.WorkbookSettings ws = new jxl.WorkbookSettings();
        ws.setEncoding("UTF-8");
        jxl.Workbook workbook = jxl.Workbook.getWorkbook(new java.io.File(path),ws);
        jxl.Sheet sheet = workbook.getSheet(1);
        for (int i = 1; i < sheet.getRows(); i++) {
                if (sheet.getCell(0, i).getContents().trim().equals("")) {
                        break;
                }
                int idSap = Integer.parseInt(sheet.getCell(0, i).getContents());
                int idSapStare = Integer.parseInt(sheet.getCell(1, i).getContents());
                String popis = sheet.getCell(2, i).getContents();
                String mernaJednotka = sheet.getCell(3, i).getContents();
                String kategorie = sheet.getCell(4, i).getContents();
                pstat.setInt(1, idSap);
                pstat.setInt(2, idSapStare);
                pstat.setString(3, popis);
                pstat.setString(4, mernaJednotka);
                pstat.setString(5, kategorie);
                citac += pstat.executeUpdate();
        }
        System.out.println("Celkem bylo do tabulky vloženo " + citac + " záznamů.");
} catch(Exception e) {
        System.out.println(e.toString());
} finally {
        ...
        pstat.close();
        con.close();
        ...
}
...
```

Příklad 4-6: Implementace knihovny JExcelAPI (naplnění tabulky *material_polozky*)

Úryvek kódu z příkladu 4-6 je obsažen v samostatné spustitelné třídě jazyka Java, která je součástí finální distribuce aplikace. Tento kód slouží k naplnění databázové tabulky *material_polozky*. Jak lze tedy interpretovat příklad 4-6? Prvních pět řádků příkazů je zaměřeno na deklaraci a inicializaci proměnných. V této části bych vyzdvihnul dvě proměnné. Jsou jimi String sqlDotaz reprezentující dotaz jazyka SQL, který se bude v programu opakovaně volat a String path představující cestu k souboru se zdrojovými daty. V bloku try konstrukce try-catch-finally nejprve díky statické metodě java.sql.Connection getConnection třídy java.sql.DriverManager inicializuji objekt con, který představuje konektivitu k databázovému serveru MySQL. Poté prostřednictvím objektu con a jeho metody java.sql.PreparedStatement prepareStatement(String sql) vytvořím instanci třídy java.sql.PreparedStatement pstat, která zastupuje parametrizovaný dotaz. Za parametr metody java.sql.PreparedStatement prepareStatement(String sql) dosadím proměnnou String sqlDotaz, o které jsem se již v tomto odstavci zmiňoval. Řetězec String sqlDotaz obsahuje několik otazníků. Každý otazník plní budoucí zástupnou funkci parametru dotazu jazyka SQL. Za otazníky můžeme dosadit různé hodnoty. Realizujeme to tak, že na objektu java.sql.PreparedStatement pstat voláme patřičnou metodu setXxx(int parameterIndex, Xxx hodnota). Například pokud chceme za druhý otazník objektu String sqlDotaz dosadit řetězec "aaa", uděláme to pomocí objektu java.sql.PreparedStatement pstat a jeho metody setString(2, "aaa"). Nyní přichází na řadu knihovna JExcelAPI. Nejprve potřebuji zajistit, aby byl soubor formátu *.xls* obsahující vstupní data načten ve správné znakové sadě. To zajistím vytvořením objektu jxl.WorkbookSettings ws, kterému metodou setEncoding(String enc) nastavím znakovou sadu na UTF-8. Poté pomocí statické metody jxl.Workbook getWorkbook(java.io.File input, jxl.WorkbookSettings ws) mohu načíst a do proměnné jxl.Workbook workbook uložit obsah souboru formátu *.xls*. Dále deklaruji a inicializuji proměnnou jxl.Sheet sheet prostřednictvím metody jxl.Sheet getSheet(int parameterIndex) objektu jxl.Workbook workbook. Objekt typu jxl.Sheet reprezentuje jeden konkrétní list sešitu. Chceme-li například získat první list celého sešitu, zadáme jako hodnotu proměnné int parametrIndex číslo 0. Po získání listu (jxl.Sheet) začíná cyklus, ve kterém se postupně po řádcích ukládají do proměnných jednotlivé buňky. Předtím, než program přejde na další řádek, dochází k parametrizování a provedení dotazu, který provede vložení záznamu do databáze MySQL. Nakonec se do konzoly vypíše celkový počet vložených záznamů.

5. Zhodnocení přínosů

V této kapitole se nejprve zaměřím na srovnání dvou zakázkových systémů, a to systému původního a systému nově vyvinutého. Poté se zmíním o možnostech budoucího rozvoje databázové aplikace a nakonec uvedu několik málo vět vyjadřujících mé osobní přínosy vzniklé během realizace diplomové práce.

Pro určení přínosů diplomové práce plynoucích pro firmu Telko s r.o. se nejdříve vrátím ke kapitole 2.3.3, kde jsem identifikoval slabé stránky původního systému zakázek. Mezi výčtem slabin původního systému byly body jako neexistence centrálního úložiště dat, výskyt duplicitních a nekonzistentních údajů, neprovázanost procesu *účtování zakázek* a procesu *vedení evidence skladu* nebo neexistence specifikace systému zakázek. Problém neexistence centrálního datového úložiště a výskytu duplicitních nebo nekonzistentních údajů řeší nový systém evidence zakázek pomocí zavedení databázového serveru MySQL. Databáze MySQL poskytuje datovou základnu pro nově vytvořenou desktopovou aplikaci napsanou v programovacím jazyce Java. Tato aplikace mimo jiné zajišťuje odstranění problému spjatého s neprovázaností procesu *účtování zakázek* a procesu *vedení evidence skladu*. Díky aplikaci je nyní možné k zakázce přiřadit zastavěný materiál a zároveň vytvořit patřičnou výdejku materiálu. Dalším přínosem pro uživatele aplikace je rychlost a jednoduchost vyhledávání a modifikace jednotlivých zakázek, možnost tiskového a elektronického výstupu vybraných formulářů a možnost škálovatelnosti aplikace.

Pokud bych měl v budoucnu ze své vlastní iniciativy nějakým způsobem aplikaci dále rozšiřovat, začal bych zřejmě u vytvoření modulu pro grafické účtování zakázek, o kterém jsem pojednal v kapitole 3.1 („Vize budoucího zakázkového systému“). Tento modul by výrazně urychlil celý proces účtování zakázek a zároveň by v pozitivním slova smyslu odlišil mou aplikaci od konkurenčních produktů. Další vyhlídkou do budoucna je možnost širšího využití systému. Za tímto účelem bych musel vyvinout obecnou aplikaci, která by byla snadno customizovatelná, spolu s případnou demonstrační verzí dostupnou volně přes internet. Dořešit by se poté ještě musela problematika přístupu více uživatelů.

Výčet osobních přínosů je dle mého názoru velmi abstraktní. Přesto se musím zmínit o zdokonalení mých dosavadních znalostí a dovedností, kterých jsem během vývoje aplikace a psaní diplomové práce nabyl. Realizace diplomové práce pro mě byla velkou zkušeností a jsem rád, že jsem dosáhl jejího konce.

6. Závěr

V úvodu diplomové práce jsem stanovil svůj cíl, kterým byla racionalizace zakázkového systému firmy Telko s r.o. Následující odstavce popisují, jakým způsobem jsem postupoval a jak jsem tento cíl realizoval.

V první kapitole jsem se nejprve zaměřil na stručný popis firmy Telko s r.o., na teoretická východiska dané problematiky a nakonec na analýzu původního zakázkového systému firmy. Na základě poznatků z této kapitoly jsem nabyl na přesvědčení, že optimální variantou racionalizace původního systému je vytvoření nového systému, který bude založen na databázovém serveru a klientské aplikaci. Druhá kapitola pak rozvíjí myšlenku vývoje nového zakázkového systému a je zaměřena na jeho návrh. Na úvod jsem v této kapitole uvedl svou vizi budoucího zakázkového systému, kterou jsem následně představil vedoucímu pracovníkovi firmy a na jejím základě byla vyhotovena specifikace uživatelských požadavků. Dále jsem se zaměřil na návrh datové struktury, na návrh a popis klíčových komponent a na návrh grafického uživatelského rozhraní klientské aplikace. Kapitolu jsem zakončil částí, která popisovala možnosti portability aplikace. Obsahem třetí kapitoly je implementace databázové aplikace. První část je věnována technologiím MySQL a Java, které jsem se rozhodnul pro vývoj zakázkového systému firmy Telko s r.o. použít. V druhé části kapitoly jsem se pak zaměřil na implementaci vybraných komponent v programovacím jazyce Java. Poslední, čtvrtá, kapitola byla věnována přínosům diplomové práce a nástinu možného budoucího rozvoje aplikace.

V době, kdy píši tento odstavec, již je systém evidence zakázek a skladu firmy Telko s r.o. v provozu. Přesto však zatím systém dle mého názoru není ve své finální podobě. Rozhodnutí o ukončení vývoje aplikace má v rukou zákazník. Díky stále přicházejícím požadavkům na přidanou funkcionalitu nebo doladění některých stávajících funkcí aplikace si troufám tvrdit, že aplikace plní svůj účel a má přínos v reálném světe.

Seznam použité literatury

Literatura a slidy

[1] COCKBURN, Alistair. *Use Cases : Jak efektivně modelovat aplikace*. Vydání první. Brno : CP Books, 2005. 262 s. ISBN 80-251-0721-3.

[2] DARWIN, Ian F. *Java: Kuchařka programátora*. Přeložil Jan Gregor. Vyd. 1. Brno : Computer Press, 2006. 799 s. ISBN 80-251-0944-5.

[3] FOWLER, Martin. *Destilované UML*. Přeložil Martin Pavlíček, redaktor Pavel Němeček. Vyd. 3. Praha : Grada Publishing, 2009. 176 s. ISBN 978-80-247-2062-3.

[4] GILMORE, W. Jason. *Velká kniha PHP 5 & MySQL : Kompendium znalostí pro začátečníky i profesionály*. Přeložil Jan Pokorný, redaktor Miroslav Kučera. Vyd. 1. Brno : Zoner Press, 2005. 711 s. ISBN 80-86815-20-X.

[5] HEROUT, Pavel. *Učebnice jazyka Java*. Vyd. 1. České Budějovice : KOPP, 2004. 349 s. ISBN 80-7232-115-3.

[6] KADLEC, Václav. *Agilní programování : Metodiky efektivního vývoje softwaru*. Vydání první. Brno : Computer Press, 2004. 278 s. ISBN 80-251-0342-0.

[7] MINISTR, Jan. *Slidy pro podporu výuky předmětu 155348 Softwarové inženýrství na EKF VŠB-TU Ostrava*. Ostrava, 2010

[8] NOVÁK, Vítězslav. *Slidy pro podporu výuky předmětu 155318 Programování IV na EKF VŠB-TU Ostrava*. Ostrava, 2008

[9] TVRDÍKOVÁ, Milena. *Aplikace moderních informačních technologií v řízení firmy : Nástroje ke zvyšování kvality informačních systémů*. Vyd. 1. [s.l.] : Grada, 2008. 176 s. ISBN 978-80-247-2728-8.

Internetové odkazy

[1i] ERD In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 15. 9. 2009, 15. 9. 2009 [cit. 2010-03-13]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/ERD>.

[2i] KOVAL, Filip. *Owebu.cz* [online]. c2010 [cit. 2010-04-06]. Java - Grafické komponenty (53.díl). Dostupné z WWW: <http://owebu.bloger.cz/Programovani/Java-Graficke-komponenty-53-dil>.

[3i] MySQL In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 8. 11. 2007, 24. 2. 2010 [cit. 2010-03-31]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/MySQL>.

[4i] Oracle. *Free and Open Source Java : Understanding the Java Platform Architecture* [online]. 2006-11-12 [cit. 2010-03-31]. Free and Open Source Java. Dostupné z WWW: <http://www.sun.com/software/opensource/java/intro_java_tech.jsp>.

[5i] SourceForge. *SourceForge.net* [online]. c2010 [cit. 2010-04-02]. Java Excel API. Dostupné z WWW: <http://jexcelapi.sourceforge.net/>.

[6i] Sun Microsystems, Inc.. Java.sun.com : The Source for Java Developers [online]. c1994-2009 [cit. 2009-11-09]. Text v angličtině. Dostupný z WWW: <http://java.sun.com/>.

[7i] Swing (Java) In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 11. 5. 2009, 5. 2. 2010 [cit. 2010-04-01]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Swing_%28Java%29>.

Seznam zkratek

API – Application programming interface

AWT – Abstract window toolkit

BIT – Binary digit

CASE – Computer-aided software engineering

CMM – Capability maturity model

CORBA – Common object request broker architecture

CRM – Customer relationship management

DSS – Decision support system

EE – Enterprise edition

EIS – Executive information system

ER – Entity-relationship

ERM – Entity-relationship model

FURPS – Functionality, usability, reliability, performance, supportability

GUI – Graphical user interface

IFC – Internet foundation classes

IS – Information system

JDBC – Java database connectivity

JFC – Java foundation classes

JRE – Java runtime enviroment

JVM – Java virtual machine

LOC – Lines of code

ME – Micro edition

OMG – Object management group

OO – Object-oriented

SE – Standard edition

SQL – Structured query language

SSADM – Structured systems analysis and design method

TPS – Transaction processing system

UC – Use case

UML – Unified modeling language

Prohlášení o využití výsledků diplomové práce

Prohlašuji, že

- byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo,
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně ke své vnitřní potřebě diplomovou práci užít (§ 35 odst. 3),
- souhlasím s tím, že jeden výtisk diplomové práce bude uložen v Ústřední knihovně VŠB-TUO k prezenčnímu nahlédnutí a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že údaje o diplomové práci, obsažené v Záznamu o závěrečné práci, umístěném v příloze mé diplomové práce, budou zveřejněny v informačním systému VŠB-TUO,
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona,
- bylo sjednáno, že užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne ……..

……..

jméno a příjmení studenta

Adresa trvalého pobytu studenta:

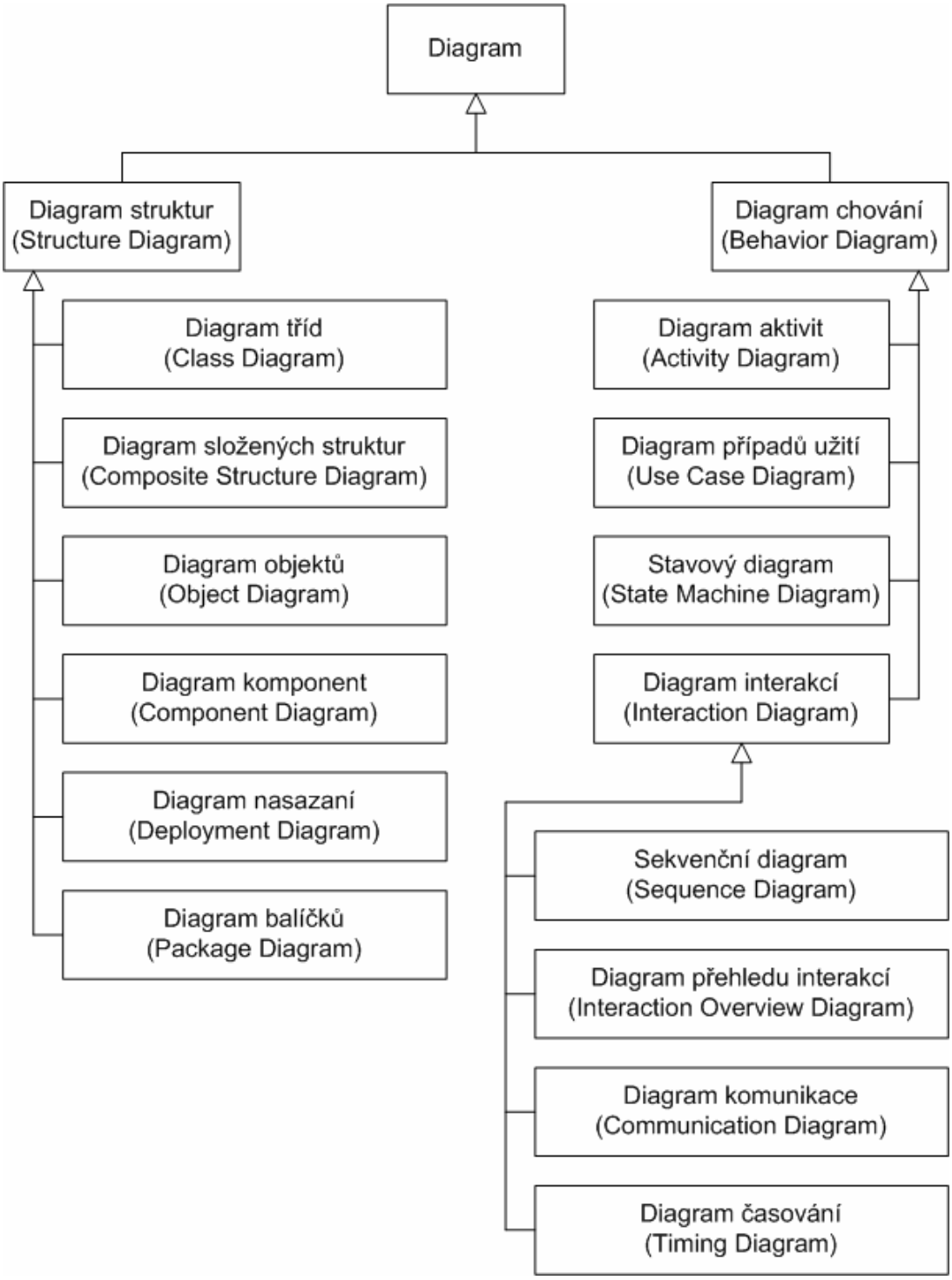
Zašová 35

Zašová

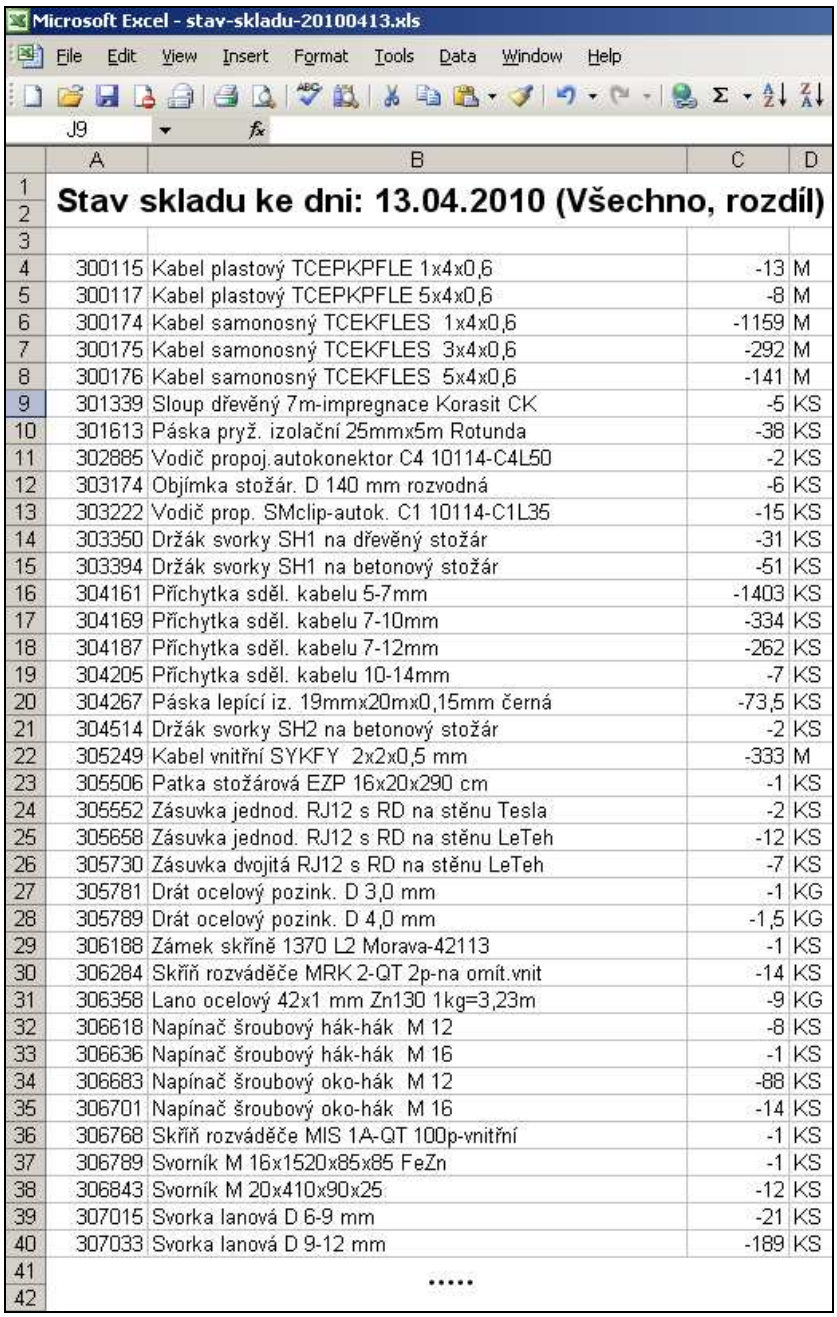
756 51

Přílohy

P1 – Členění UML diagramů



P2 – Ukázka výstupu do formátu *.xls* (stav skladu)



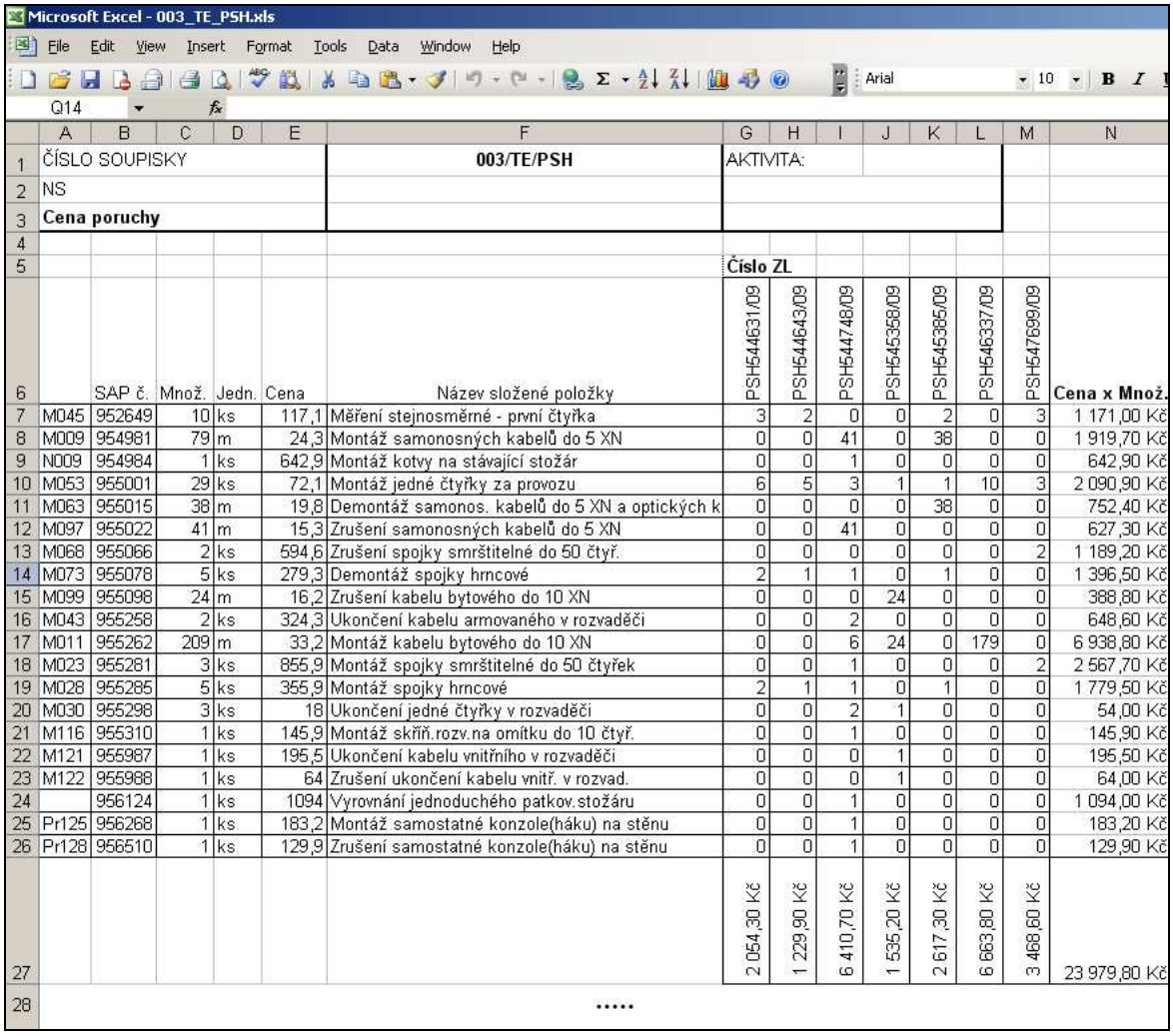
Microsoft Excel - stav-skladu-20100413.xls

File Edit View Insert Format Tools Data Window Help

J9

	A	B	C	D
1	**Stav skladu ke dni: 13.04.2010 (Všechno, rozdíl)**			
2				
3				
4	300115	Kabel plastový TCEPKPFLE 1x4x0,6	-13	M
5	300117	Kabel plastový TCEPKPFLE 5x4x0,6	-8	M
6	300174	Kabel samonosný TCEKFLES 1x4x0,6	-1159	M
7	300175	Kabel samonosný TCEKFLES 3x4x0,6	-292	M
8	300176	Kabel samonosný TCEKFLES 5x4x0,6	-141	M
9	301339	Sloup dřevěný 7m-impregnace Korasit CK	-5	KS
10	301613	Páska pryž. izolační 25mmx5m Rotunda	-38	KS
11	302885	Vodič propoj.autokonektor C4 10114-C4L50	-2	KS
12	303174	Objímka stožár. D 140 mm rozvodná	-6	KS
13	303222	Vodič prop. SMclip-autok. C1 10114-C1L35	-15	KS
14	303350	Držák svorky SH1 na dřevěný stožár	-31	KS
15	303394	Držák svorky SH1 na betonový stožár	-51	KS
16	304161	Příchytka sděl. kabelu 5-7mm	-1403	KS
17	304169	Příchytka sděl. kabelu 7-10mm	-334	KS
18	304187	Příchytka sděl. kabelu 7-12mm	-262	KS
19	304205	Příchytka sděl. kabelu 10-14mm	-7	KS
20	304267	Páska lepící iz. 19mmx20mx0,15mm černá	-73,5	KS
21	304514	Držák svorky SH2 na betonový stožár	-2	KS
22	305249	Kabel vnitřní SYKFY 2x2x0,5 mm	-333	M
23	305506	Patka stožárová EZP 16x20x290 cm	-1	KS
24	305552	Zásuvka jednod. RJ12 s RD na stěnu Tesla	-2	KS
25	305658	Zásuvka jednod. RJ12 s RD na stěnu LeTeh	-12	KS
26	305730	Zásuvka dvojitá RJ12 s RD na stěnu LeTeh	-7	KS
27	305781	Drát ocelový pozink. D 3,0 mm	-1	KG
28	305789	Drát ocelový pozink. D 4,0 mm	-1,5	KG
29	306188	Zámek skříně 1370 L2 Morava-42113	-1	KS
30	306284	Skříň rozváděče MRK 2-QT 2p-na omít.vnit	-14	KS
31	306358	Lano ocelový 42x1 mm Zn130 1kg=3,23m	-9	KG
32	306618	Napínač šroubový hák-hák M 12	-8	KS
33	306636	Napínač šroubový hák-hák M 16	-1	KS
34	306683	Napínač šroubový oko-hák M 12	-88	KS
35	306701	Napínač šroubový oko-hák M 16	-14	KS
36	306768	Skříň rozváděče MIS 1A-QT 100p-vnitřní	-1	KS
37	306789	Svorník M 16x1520x85x85 FeZn	-1	KS
38	306843	Svorník M 20x410x90x25	-12	KS
39	307015	Svorka lanová D 6-9 mm	-21	KS
40	307033	Svorka lanová D 9-12 mm	-189	KS
41	.....			
42				

P3 – Ukázka výstupu do formátu *.xls* (souhrnné vyúčtování – list 1)

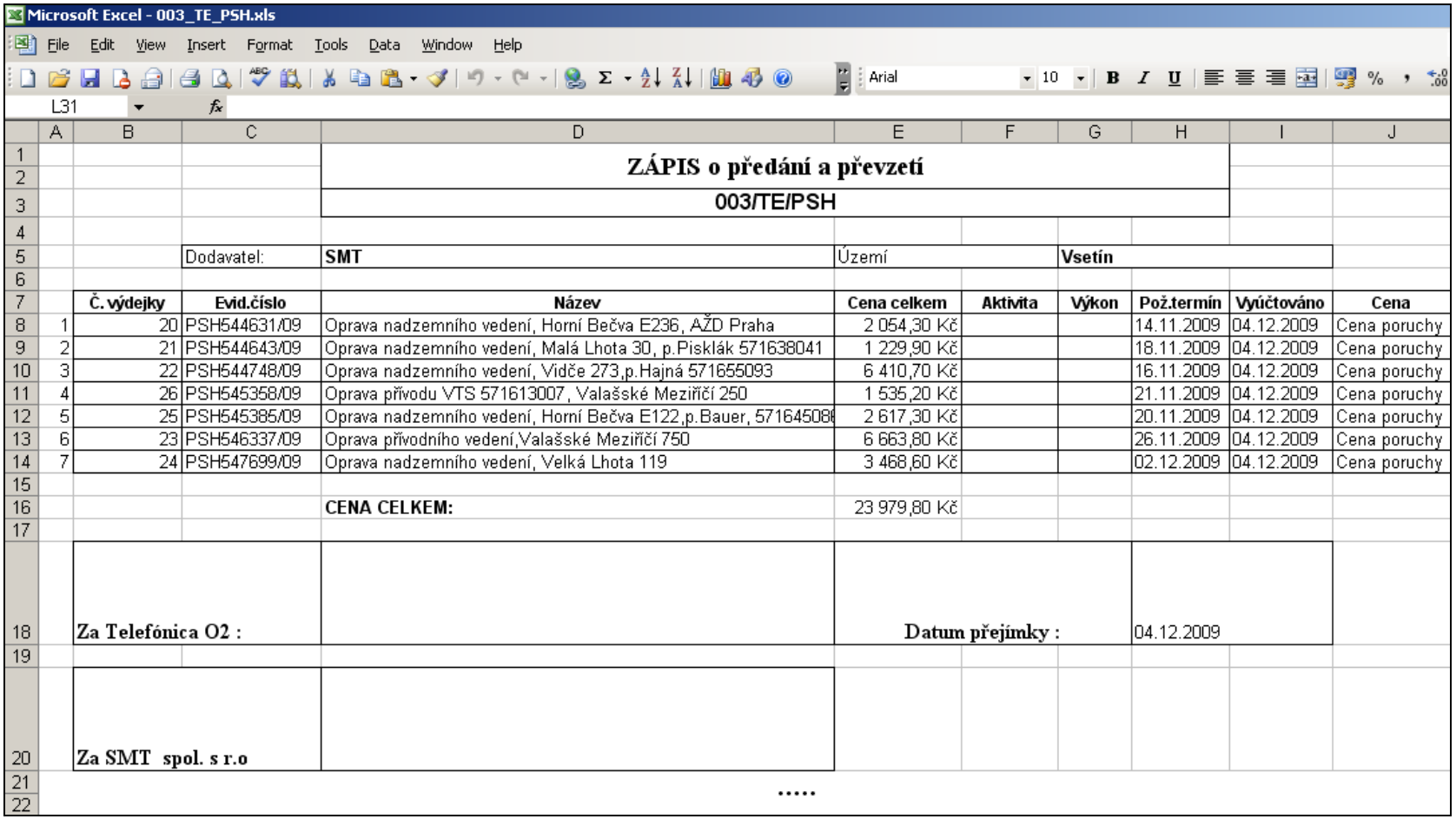


Microsoft Excel - 003_TE_PSH.xls

File Edit View Insert Format Tools Data Window Help

Q14

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	ČÍSLO SOUPISKY					**003/TE/PSH**	AKTIVITA:							
2	NS													
3	**Cena poruchy**													
4														
5							**Číslo ZL**							
6		SAP č.	Množ.	Jedn.	Cena	Název složené položky	PSH544631/09	PSH544643/09	PSH544748/09	PSH545358/09	PSH545385/09	PSH546337/09	PSH547699/09	**Cena x Množ.**
7	M045	952649	10	ks	117,1	Měření stejnosměrné - první čtyřka	3	2	0	0	2	0	3	1 171,00 Kč
8	M009	954981	79	m	24,3	Montáž samonosných kabelů do 5 XN	0	0	41	0	38	0	0	1 919,70 Kč
9	N009	954984	1	ks	642,9	Montáž kotvy na stávající stožár	0	0	1	0	0	0	0	642,90 Kč
10	M053	955001	29	ks	72,1	Montáž jedné čtyřky za provozu	6	5	3	1	1	10	3	2 090,90 Kč
11	M063	955015	38	m	19,8	Demontáž samonos. kabelů do 5 XN a optických k	0	0	0	0	38	0	0	752,40 Kč
12	M097	955022	41	m	15,3	Zrušení samonosných kabelů do 5 XN	0	0	41	0	0	0	0	627,30 Kč
13	M068	955066	2	ks	594,6	Zrušení spojky smrštitelné do 50 čtyř.	0	0	0	0	0	0	2	1 189,20 Kč
14	M073	955078	5	ks	279,3	Demontáž spojky hrncové	2	1	1	0	1	0	0	1 396,50 Kč
15	M099	955098	24	m	16,2	Zrušení kabelu bytového do 10 XN	0	0	0	24	0	0	0	388,80 Kč
16	M043	955258	2	ks	324,3	Ukončení kabelu armovaného v rozvaděči	0	0	2	0	0	0	0	648,60 Kč
17	M011	955262	209	m	33,2	Montáž kabelu bytového do 10 XN	0	0	6	24	0	179	0	6 938,80 Kč
18	M023	955281	3	ks	855,9	Montáž spojky smrštitelné do 50 čtyřek	0	0	1	0	0	0	2	2 567,70 Kč
19	M028	955285	5	ks	355,9	Montáž spojky hrncové	2	1	1	0	1	0	0	1 779,50 Kč
20	M030	955298	3	ks	18	Ukončení jedné čtyřky v rozvaděči	0	0	2	1	0	0	0	54,00 Kč
21	M116	955310	1	ks	145,9	Montáž skříň.rozv.na omítku do 10 čtyř.	0	0	1	0	0	0	0	145,90 Kč
22	M121	955987	1	ks	195,5	Ukončení kabelu vnitřního v rozvaděči	0	0	0	1	0	0	0	195,50 Kč
23	M122	955988	1	ks	64	Zrušení ukončení kabelu vnitř. v rozvad.	0	0	0	1	0	0	0	64,00 Kč
24		956124	1	ks	1094	Vyrovnání jednoduchého patkov.stožáru	0	0	1	0	0	0	0	1 094,00 Kč
25	Pr125	956268	1	ks	183,2	Montáž samostatné konzole(háku) na stěnu	0	0	1	0	0	0	0	183,20 Kč
26	Pr128	956510	1	ks	129,9	Zrušení samostatné konzole(háku) na stěnu	0	0	1	0	0	0	0	129,90 Kč
27							2 054,30 Kč	1 229,90 Kč	6 410,70 Kč	1 535,20 Kč	2 617,30 Kč	6 663,80 Kč	3 468,60 Kč	23 979,80 Kč
28	.....													



Microsoft Excel - 003_TE_PSH.xls

File Edit View Insert Format Tools Data Window Help

L31

ZÁPIS o předání a převzetí

003/TE/PSH

Dodavatel: **SMT** Území **Vsetín**

	Č. výdejky	Evid.číslo	Název	Cena celkem	Aktivita	Výkon	Pož.termín	Vyúčtováno	Cena
1	20	PSH544631/09	Oprava nadzemního vedení, Horní Bečva E236, AŽD Praha	2 054,30 Kč			14.11.2009	04.12.2009	Cena poruchy
2	21	PSH544643/09	Oprava nadzemního vedení, Malá Lhota 30, p.Pisklák 571638041	1 229,90 Kč			18.11.2009	04.12.2009	Cena poruchy
3	22	PSH544748/09	Oprava nadzemního vedení, Vidče 273,p.Hajná 571655093	6 410,70 Kč			16.11.2009	04.12.2009	Cena poruchy
4	26	PSH545358/09	Oprava přívodu VTS 571613007, Valašské Meziříčí 250	1 535,20 Kč			21.11.2009	04.12.2009	Cena poruchy
5	25	PSH545385/09	Oprava nadzemního vedení, Horní Bečva E122,p.Bauer, 57164508	2 617,30 Kč			20.11.2009	04.12.2009	Cena poruchy
6	23	PSH546337/09	Oprava přívodního vedení,Valašské Meziříčí 750	6 663,80 Kč			26.11.2009	04.12.2009	Cena poruchy
7	24	PSH547699/09	Oprava nadzemního vedení, Velká Lhota 119	3 468,60 Kč			02.12.2009	04.12.2009	Cena poruchy

CENA CELKEM: 23 979,80 Kč

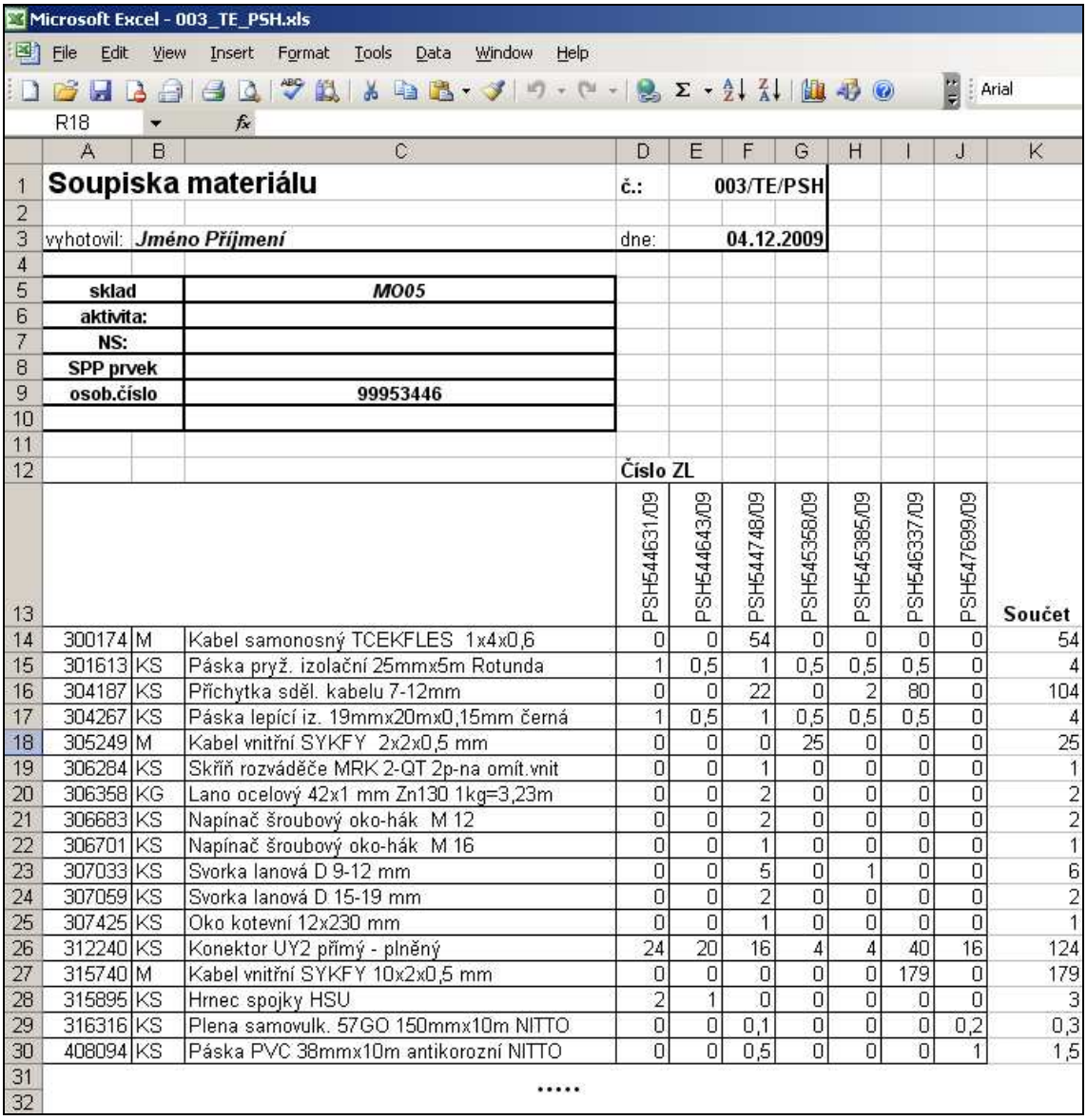
Za Telefónica O2 : Datum přejímky : 04.12.2009

Za SMT spol. s r.o

.....

P4 – Ukázka výstupu do formátu *.xls* (souhrnné vyúčtování – list 2)

P5 – Ukázka výstupu do formátu *.xls* (souhrnné vyúčtování – list 3)



Microsoft Excel - 003_TE_PSH.xls

File Edit View Insert Format Tools Data Window Help

R18

	A	B	C	D	E	F	G	H	I	J	K
1	**Soupiska materiálu**			**č.:**		**003/TE/PSH**					
2											
3	vyhotovil:	***Jméno Příjmení***		dne:		**04.12.2009**					
4											
5	**sklad**		***MO05***								
6	**aktivita:**										
7	**NS:**										
8	**SPP prvek**										
9	**osob.číslo**		**99953446**								
10											
11											
12				**Číslo ZL**							
13				PSH544631/09	PSH544643/09	PSH544748/09	PSH545358/09	PSH545385/09	PSH546337/09	PSH547699/09	**Součet**
14	300174	M	Kabel samonosný TCEKFLES 1x4x0,6	0	0	54	0	0	0	0	54
15	301613	KS	Páska pryž. izolační 25mmx5m Rotunda	1	0,5	1	0,5	0,5	0,5	0	4
16	304187	KS	Příchytka sděl. kabelu 7-12mm	0	0	22	0	2	80	0	104
17	304267	KS	Páska lepící iz. 19mmx20mx0,15mm černá	1	0,5	1	0,5	0,5	0,5	0	4
18	305249	M	Kabel vnitřní SYKFY 2x2x0,5 mm	0	0	0	25	0	0	0	25
19	306284	KS	Skříň rozváděče MRK 2-QT 2p-na omít.vnit	0	0	1	0	0	0	0	1
20	306358	KG	Lano ocelový 42x1 mm Zn130 1kg=3,23m	0	0	2	0	0	0	0	2
21	306683	KS	Napínač šroubový oko-hák M 12	0	0	2	0	0	0	0	2
22	306701	KS	Napínač šroubový oko-hák M 16	0	0	1	0	0	0	0	1
23	307033	KS	Svorka lanová D 9-12 mm	0	0	5	0	1	0	0	6
24	307059	KS	Svorka lanová D 15-19 mm	0	0	2	0	0	0	0	2
25	307425	KS	Oko kotevní 12x230 mm	0	0	1	0	0	0	0	1
26	312240	KS	Konektor UY2 přímý - plněný	24	20	16	4	4	40	16	124
27	315740	M	Kabel vnitřní SYKFY 10x2x0,5 mm	0	0	0	0	0	179	0	179
28	315895	KS	Hrnec spojky HSU	2	1	0	0	0	0	0	3
29	316316	KS	Plena samovulk. 57GO 150mmx10m NITTO	0	0	0,1	0	0	0	0,2	0,3
30	408094	KS	Páska PVC 38mmx10m antikorozní NITTO	0	0	0,5	0	0	0	1	1,5
31			.....								
32											